IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Serial No.:	09/385,394	Confirmation No.:	unassigned
Appellant:	John S. Yates, Jr., et al.		
Title:	COMPUTER WITH TWO EXECUTION MODES		
Filed:	August 30, 1999		
Art Unit:	2183		
Examiner:	Richard Ellis		
Atty. Docket:	114596-03-4000		
Customer No.	38492		

RESPONSE TO OFFICE ACTION, OR IN THE ALTERNATIVE, APPEAL BRIEF

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

In the event that the currently-pending Petition for Withdrawal of Finality is granted, Appellant hereby submits this paper as a request for reconsideration of the Office Action of 10/25/2005. If the Petition is granted, then the Notice of Appeal filed March 28, 2005 is nugatory.

In the event that all grounds petitioned, after full decision, are denied, then in the alternative, Appellant hereby submits this Appeal Brief in response to the Office Action of October 25, 2004, and new grounds raised in the Advisory Action of 2/14/05. In that event, kindly charge the fee for an Appeal Brief ($500.00) to Deposit Account No. 23-2405, Order No. 114596-03-4000.

INTRODUCTION

This application is not in condition for appeal. Twice, the Examiner has played an aggressive game of "hide the ball," keeping his position secret until post-final advisory actions.

I certify that this correspondence, along with any documents referred to therein, is being deposited with the United States Postal Service on November 28, 2005 as First Class Mail in an envelope with sufficient postage addressed to Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

While the Examiner's position on some claims and issues is now clear enough to permit briefing and adjudication, on others, the Examiner has expressed no view at all.

The PTO has caused this procedural train wreck by failing to follow or enforce its own procedural regulations. In a number of telephone calls and formal petitions, Appellant has sought the assistance of those charged by statute to "cause an examination to be made." 35 U.S.C. § 131. They have declined to even review the issues. T.C. Director's papers of 5/22/2003 and 11/8/2005 (both "dismissing," not "denying," all issues relating to "timely and complete" action by the Examiner); Summary of Interview (10/30/2005) With T.C. Director Harvey (T.C. Director refuses to acknowledge the force of court and agency precedent, and declines to adjudicate issues squarely presented).

Thus, this application reaches appeal in a state analogous to previous cases where the Board finds itself unable to adjudicate. *Ex parte Rozzi*, 63 USPQ2d 1196, 1200 (BPAI 2002) (unpublished) (declining to adjudicate: "The examiner makes no cogent attempt to read Hill onto claim 1."); *Ex parte Gambogi*, 62 USPQ2d 1209, 1212 (BPAI 2001) (unpublished) (declining to "speculate" at a position that should be stated in first instance by the examiner); *Ex parte Schricker*, 56 USPQ2d 1723, 1725 (BPAI 2000) (unpublished) ("The examiner has left and the board to guess as to the basis of the rejection ... We are not good at guessing; hence, we decline to guess."); *Ex parte Braeken*, 54 USPQ2d 1110, 1113 (BPAI 1999) (declining to adjudicate: examiner's explanation is "manifestly not ready for a decision," the appeal is "not ripe.").

Appellant fully recognizes that many of the issues presented here are not within the jurisdiction of the Board of Patent Appeals and Interferences under 35 U.S.C. § 134. However, when presented by Petition, the Director refused to consider them, insisting that the issues are appealable despite the Board's own precedent to the contrary. T.C. Director's Decision of 5/22/2003 at page 5. Thus, issues that have not been adjudicated (or that were not presented when their presentation would be futile) are now presented to the Board. Appellant requests a clear, precedential statement that certain issues are not appealable, to relieve the jurisdictional confusion that prevails in the examining operation.

TABLE OF CONTENTS

I. Real Party in Interest

The real party in interest is ATI International SRL of Barbados, the assignee of this application. ATI International is related to ATI Technologies, Inc. of Ontario, Canada.

II. Related Appeals and Interferences

Appellant is unaware of any related appeals or interferences.

III. Status of Claims

Claims 1-60, 63-115, and 118-141 are now pending, a total of 137 claims. Claims 22-36, 51-60, 63-112 and 134-141 are not allowed; however as noted below, many of them have not been examined in conformance with PTO and Federal Circuit procedural minima, and are therefore not rejected, either.

Of the non-allowed claims, claims 22, 51, 63, 87, 94, 96, 104 and 136 are independent.

A complete copy of the claims is attached hereto as an Appendix.

IV. Status of Amendments

The Amendment of 4/25/2005 is entered by operation of law because the PTO has waived all procedural rules in this application, and has waived any rights it may have had to exclude the amendment under 37 C.F.R. § 1.116(b)(2) and (3).

The fees for claims added by the Amendment of April 25, 2005 were charged to the Deposit Account. This, in combination with the following showings, demonstrates that the Amendment was entered.

The following issues are within supplementary jurisdiction of the Board, under *In re Searles*, 422 F.2d 431, 435, 164 USPQ 623, 626 (CCPA 1970) (an issue "clearly ... 'determinative of the rejection,'" that is ordinarily reviewable "only" by petition, is concurrently reviewable by the Board when it "additionally, and necessarily, required the exercise of technical skill and legal judgment").

A. The 4/25/2005 Amendment May Be Entered Pursuant to the Suspension of the Rules By T.C. Directors Levy and Harvey

The T.C. Director expressly stated that procedural law, for example, the precedent of the PTO's reviewing courts, will not be applied to this application. Papers of 9/9/05 and 11/8/05 ("Contrary to the citations of case law... it cannot be seen...," quoting the Federal Circuit's and CCPA's test for "new ground of rejection," and stating that it would not be applied).

The Patent Office cannot pick and choose which rules will be enforced and which will be waived. *E.g.*, Paper of 11/8/05, ignoring 37 C.F.R. §§ 1.104, 1.113 and MPEP § 706.07, the rules under with relief was sought, and instead relying on a half sentence from MPEP § 706.07(a). The T.C. Director states that he will not make an appropriate inquiry to determine what the law is, let alone apply it. Summary of Interview (10/30/2005) With T.C. Director Harvey ¶ 4 (expressly refusing to inquire beyond the MPEP, and declining an express offer of legal research from which the law can be determined). The Patent Office has also waived any limits on introducing new grounds of rejection in a final Office Action or in post-final Advisory Actions, even when the Advisory Action itself expressly admits that all prior papers have been incomplete and unclear. *Compare* Advisory Action of 2/14/2005, paragraph spanning pages 1-2 (admitting that all prior papers had failed to consider an issue, and citing a new portion of a reference), Supplement to Petition of 6/14/2005 *and* Summary of Interview (10/30/2005) *and* T.C. Director Harvey's paper of 11/8/05 (in some cases, omitting any consideration of whether other present Office practices were timely and completely followed, in other cases expressly acknowledging that rules were breached, but examiner would be permitted to cure out of time).

The PTO is required to apply its rules "in a fair, impartial, and equitable manner." 35 U.S.C. § 3(a)(2)(A). Because the PTO cannot unilaterally waive only the rules that apply to itself, or create new exceptions to existing rules on the fly, the waiver of rules announced in the two papers of the T.C. Directors must have been bilateral.

Therefore, rules that impose procedural obligations and time deadlines on applicants are also waived by operation of law. Any rules that would deny entry to the Amendment of 4/25/2005 have been waived.

B. The Office Failed to Act to Exclude the Amendments Under Rule 116

The "Amendment Accompanying Notice of Appeal" of 4/25/05 includes showings that the amendment qualifies under 37 C.F.R. § 1.116(b)(2) or (b)(3). For example, the amendment to claim 63 is properly entered under Rule 116(b)(3) because it could not have been presented earlier than the new ground of rejection stated by the Examiner in the Advisory Action of 2/14/05, and is necessary to respond thereto.

Neither the Examiner nor the T.C. Director have ever responded to any issue relating to Rule 116(b)(3). The Examiner, for example, only comments in his Advisory of 6/7/05 that "New language added to claim 63 would require further search and consideration." The T.C. Director's paper of 11/08/2005 only comments that claim 63 "requires more than a cursory review." Perhaps these observations might be relevant under Rule 116(b)(1), but they are totally irrelevant to Rule 116(b)(3). Neither the Examiner nor the T.C. Director contest that there is anything other than "good and sufficient reasons why the amendment is necessary and was not earlier presented."

There being no dispute on the facts, there is no basis to refuse entry.

Further, because the PTO's time has run out to adjudicate the precise issues presented to it, 5 U.S.C. § 555(a) ("With due regard for the ... necessity of the parties ... and within a reasonable time, each agency shall proceed to conclude a matter presented to it."), the 4/25/2005 amendment may not now be refused.

C. Denial of Entry of the Amendment of 4/25/2005 Was *Ultra Vires* Any Authority of the Examiner, and is Therefore Void

At paragraph 4 of the Advisory Action of 2/14/2005, in the paragraph spanning pages 1-2, the Examiner expressly admits that his previous office actions were neither "clear" nor "complete." By the Examiner's own admission, he did not satisfy the prerequisites of § 1.113 to impose final rejection, or deny entry of the amendment of 4/25/2005. Because finality is premature, the Amendment of 4/25/2005 is entered.

V. Informal Summary of the Subject Matter

Generally, and without prejudice to the scope of the claims, this § V presents a primer in the technology, and presents a high level view of the specification.[1]

Generally, the disclosure relates to a single computer processor that is capable of running two different instruction sets. In the example of the specification, the two instruction sets are the X86 (now more commonly known as "Pentium" or "IA-32," the traditional line of computers from Intel Corp.), and Tapestry, a RISC processor tailored for high-performance graphics, while providing reasonable execution of X86 programs. The specification discusses a single computer that can run both off-the-shelf programs for the Intel X86, and programs specialized and optimized for the Tapestry RISC processor.

Supporting multiple instruction sets may involve, among other things:

- supporting calls and transfers between instruction sets. For example, a computer might provide a high-performance graphics or image-processing library in a RISC instruction set, and allow off-the-shelf programs coded in the X86 instruction set to call these RISC libraries.
- supporting data types that are specific to one computer but not directly implemented on the other. For example, addresses in the Intel processor are 16 bits, 32 bits or 48 bits wide, but in the Tapestry processor, addresses are 64 bits wide.
- supporting an operating system, for example, Microsoft Windows or UNIX, that is not native to the computer. For example, an operating system such as Microsoft Windows that knows only how to manage X86 hardware resources might be made useable on RISC hardware for which that operating system software has not been tailored.

Sections I-IV of the specification (pages 31-60) discuss mechanisms for solving some of these problems. An introductory overview of preferred embodiments appears at pp. 32-33 of the specification.[2]

---

[1] The discussion here, and the discussion of the specification in the arguments below in connection with certain claims, is merely offered as a concrete preferred embodiment to provide context for understanding of the claims. Such introduction to the technology for members of the Board, who may not be specialists in this technological field, is not a limiting description of the invention or any claim.

[2] This § V is a mere discussion of one concrete preferred embodiment, as a helpful aid to establish context for the discussion of the claims that follows. It is not a discussion of the invention or the claims themselves.

Figs. 1a, 1c, 2a, 2b and 2c, 3a-3o and §§ II, III and IV (pp. 51-54 and 66-75) discuss different regions of code of a program that embody different instruction sets and/or data storage conventions.

Instruction sets, and the differences among them, are explained at pages 1-2 of the specification. For example, one of the key differences between Apple McIntosh computers and Windows/Intel PC's are the different ways instructions are represented in the two families of computers.

Another difference among computers is "data storage conventions." In order to permit interoperability of software components, various people and companies agree among each other that certain data will be stored in certain locations. One example of a "data storage convention" is a "calling convention," an agreement among compiler writers, operating system programmers, programmers of other various software tools, etc. that certain data will be stored in certain memory locations or registers.[3] Calling conventions are fundamentally purely agreements among people and the software they create. Thus, even where some hardware assistance may exist, programmers are free to "hand tailor" the storage of their data without observing the calling convention, or to otherwise modify the calling convention. For example, different segments of the Windows/Intel community use somewhat different calling conventions.

On the other hand, some data storage conventions are inextricably defined into hardware. For example, the representation of floating-point numbers (numbers with digits to the right of the decimal point, as opposed to integers) are defined into the hardware definition of their respective instruction sets. It would be impractical to process floating-point numbers in the radix-16 format used in IBM mainframes, the F- and G-formats used in Digital's VAX computers, or the 80-bit format used in Intel's X86 floating-point registers, on computers that lack hardware for the particular format.

From time to time, it becomes necessary or desirable to combine program code that uses one instruction set or data storage convention with code from another instruction set or data storage convention. For example, Microsoft does not make its Windows source code available to

---

[3] Calling conventions are discussed in detail in §§ III.A, III.B, and IV of the specification, most particularly at pages 51-52.

be tailored to new kinds of computers, and thus non-traditional methods are required to make the rich Windows environment available on non-Intel computers.

One known mechanism is to provide a software "simulator" (specification, page 2, lines 11-17), a software program that creates a software "virtual" copy of the foreign CPU, which can execute the foreign code (though typically at a 10X or more slowdown). If it is desired to "mix and match" code for the foreign CPU (e.g., to test the code or the CPU design) with "native" code (that executes directly on the hardware, and therefore executes much faster than a simulator), one known approach is to interpose a software "jacket" at the boundaries between the code for the old computer and the code for the new computer. A "jacket" is somewhat analogous to a "pipe adapter" for joining two different diameter or differently-threaded pipes – the "jacket" is a bit of program code precisely tailored to form a "joint" between two pieces of a program. Just as a pipe adapter must have two very specific diameters and threadings, tailored to the two precise pipe ends to be joined, a software "jacket" is very precisely tailored to one entry or exit point from the mismatched code. Instructions that transfer control from old code to new, or vice-versa, must be "patched" so that they transfer control to the jacket, not directly to the destination. The jacket makes appropriate adjustments, and then the jacket transfers control to the destination.[4] If this "patched" code was intended to be "self-modifying," or if certain features such as stack traceback or an exception handler rely on particular data contents in particular locations, then patching or the indirect transfer through the jacket can prevent correct execution.

In contrast, the specification and drawings teach several approaches that may be used together or separately to provide a more-general and less-invasive approach. Regions of code are marked with ISA (instruction set architecture) bits 180, 182, 194 and "calling convention" bits 184, 186, 196, 200 in Figs. 1a, 1d and 2a that indicate the instruction set and/or calling convention used by the respective regions. Each region of code may execute more or less as it would on a "normal" computer for that specific instruction set and/or data storage convention. However, when control transfers to a destination with an indicator element indicating a different instruction set and/or data storage convention, then the system notices the transition. The system

[4] The use of jackets or analogous techniques is not disclaimed – use of jackets with the claimed subject matter still falls within the claims. This is merely a background discussion.

automatically makes whatever modifications are necessary (Fig. 2b, Figs. 3a-3o) before transferring control to the destination, and allowing execution to resume.

For example, if the transition relates to instruction set architecture, the instruction decoder is placed in a mode to decode the new instruction.

If the transition involves a data storage convention, the content of storage of the computer is adjusted so that the code at the destination sees a situation that appears, and may execute more or less as if, this were a "normal" computer for the new mode. For example, data might be copied from particular registers to particular locations in memory, as dictated by the differing assumptions, and/or the format may be changed from the format demanded by the source mode to the format demanded by the destination mode.

Figs. 2a-2c and 3a-3o and the specification (discussed in §§ II and III, pages 44-65) discuss a computer that has two reasonably "pure bred" modes, in which the computer follows first and second instruction sets and their respective data storage conventions, and a "mixed" mode in which the data storage convention of one is paired with the instruction set of the other. When execution flows from one region to the other, the computer adapts itself accordingly. For example, cols. 194 and 200 of Fig. 2a show these three modes:

- the first row of Fig. 2a shows a mode in which the processor executes the Tapestry RISC instruction set, using Tapestry RISC calling conventions (a "calling convention" is one type of "data storage convention")
- the third row of Fig. 2a shows a mode in which the processor executes the X86 instruction set, using X86 calling conventions
- the second row of Fig. 2a shows a "mixed mode" in which the processor executes the Tapestry RISC instruction set, but uses X86 calling conventions

The process of adaptation between the modes is shown in Figs. 1a, 1c, and 3a-3o, and discussed in §§ II and III (pages 44-65).

Thus, in a preferred implementation, complex X86 programs such as Microsoft Windows or application programs can execute on a computer that also provides the capability of executing RISC instructions that may provide very fast execution of certain programs, even if the source code for the program is not available for recompilation or other translation into a new form. The combined execution modes may be better than either alone.

Certain "bookkeeping" is advantageously performed on the basis of physical addresses. Some of the ISA bits or "calling convention" bits of the specification are stored in tables whose entries are indexed by physical page frame number, instead of in the more-conventional approach, tables indexed by virtual address. Two such tables and their entries are shown as PFAT 172 in Fig. 1d and PIPM 602 of Fig. 6a. To locate the relevant entry of PFAT 172 or PIPM 602, an address typically undergoes logical-to-physical address translation (*e.g.*, using the "page directory," "page tables," and/or "TLB" of at the left half of Fig. 1d). Then, that physical address is used to index into the PFAT or PIPM to locate a particular entry.

The specification teaches a mechanism for allowing control-transfer instructions to transfer control from a source to a destination, even if the destination is in a mismatched mode, in a way that reduces the perturbation of the environment at both the source and destination. For example, the "call" instruction 383 of Fig. 3m appears to be an ordinary X86 call instruction that transfers control to destination 317, 319. Call instruction 383 may use "PC+displacement" addressing, or the address of destination 317, 319 may be in a register, as if it were a "normal" X86 call to the desired destination. However, when call instruction 383 executes to transfer to RISC portion 308, the system detects the change of execution mode ("flash" 384). In response, the system executes the "X86->RISC" transition handler 320. Handler 320 moves data from its normal X86 "home" to its RISC "home," so that the RISC code 308 can execute. The mechanism here contrasts to the "jacket" mechanism discussed above, where the call instruction must be altered either to call to the jacket (instead of to the desired destination), or altered so that it faults, and does not execute at all, so that execution will be forced to go through the jacketing process.

VI. Argument

A. Standard of Review

On issues of claim construction, fact, and law, the Board reviews *de novo*, that is, with no deference. *E.g.*, *Ex parte Toda*, App. No. 98-0078, 2001 WL 1729659 at *3, www.uspto.gov/web/offices/dcom/bpai/decisions/fd980078.pdf at 5 (BPAI Apr. 26, 2001).

The Supreme Court instructs that an agency action taken in violation of "applicable departmental regulations" is "illegal and of no effect." *Accardi v. Shaughnessey*, 359 U.S. 535, 545 (1959); *Schroeder v. West*, 212 F.3d 1265, 1269-70 (Fed. Cir. 2000). This Board cannot affirm a rejection when the proceedings during the examination phase have been inadequate. *Stone v. Federal Deposit Insurance Corp.*, 179 F.3d 1368, 1376 (Fed. Cir. 1999) ("Our system is premised on the procedural fairness at each stage of [agency] proceedings. [A party before an agency] is entitled to a certain amount of due process rights at each stage and, when those rights are undermined, the [party] is entitled to relief regardless of the stage of the proceedings."). The Board cannot affirm a rejection that was raised in violation of agency procedures, because no rejection exists.

B. Claim 22

The Examiner's current view of claim 22 cannot be determined. Because there is no "clear explanation" of any rejection, as required by 37 C.F.R. § 1.104(c)(2), claim 22 is not rejected.

Claim 22 recites as follows:

> 22. A method, comprising the steps of:
>
> executing instructions fetched from first and second regions of a memory of a computer, the instructions of the first and second regions being coded for execution by computers following first and second data storage conventions, the memory regions having associated first and second indicator elements, **the indicator elements each having a value indicating the data storage convention under which instructions from the associated region are to be executed**;
>
> recognizing when program execution has flowed or transferred from a region whose indicator element indicates the first data storage convention to a region whose indictor element indicates the second data storage convention, and in response to the recognition, altering the data storage content of the computer to create a program context under the second data storage convention that is logically equivalent to a pre-alteration program context under the first data storage convention.

1. Procedural History of Claim 22

Appellant has responded to the issues "in the prior Office action," 37 C.F.R. § 1.111(b), and has made a good faith attempt to guess at and respond to those points where the Examiner

has been silent. See papers of January 2005 and July 2004. However, until the Examiner puts his reasoning on paper in the manner required by law, (a) as a matter of administrative law, the claims are not rejected, and (b) Appellant cannot meaningfully identify any difference of opinion for appeal, let alone frame that appeal.

Giving ¶¶ 52.3 and 52.4 of the February 2004 Action their most generous reading, the Office Action compared the "the indicator elements each having a value indicating the data storage convention under which instructions from the associated region are to be executed" to only Goetz '913, Fig. 10, col. 14, lines 18-22, and col. 17, lines 24-33 and to no other reference. (¶¶ 52.1, 52.2 and 52.5 also only mention the Goetz '913 reference, making clear that one reference is relied upon exclusively for this portion of the claim):

> **52. As to claim 22, Goetz et al. in view of Brender et al. and Murphy et al. taught:**
>
> **52.3. the memory regions having associated first and second indicator elements (Goetz et al. at fig. 10 and col. 14 lines 18-22),**
>
> **52.4. the indicator elements each having a value indicating the data storage convention under which instructions from the associated region are to be executed (Goetz et al. col. 17 lines 24-33);**

Appellant's paper of July 2004 fully responded to the issue as presented in the Office Action. The July 2004 paper showed that the Examiner's interpretation of Goetz '913, col. 17, lines 24-33 is incorrect, and that the claim limitations discussed in ¶¶ 52.3 and 52.4 are note met by Goetz '913.

The October 2004 Action (¶ 14.1) hints that the Examiner believes that other references are pertinent to the particular claim language addressed in ¶¶ 52.3 and 52.4 of the February 2004 Action. However, the October 2004 Office Action does not even specify which other references might be involved, let alone "designate the portions relied on" or "clearly explain" their pertinence:

14.1. That: "Paragraph 52.4 compares the "the indicator elements each having a value indicating the data storage convention under which instructions from the associated region are to be executed" to Goetz, col. 17, lines 24-33. However, here, Goetz only teaches a P bit that indicates an instruction set. There is no indication that Goetz ever uses two different "data storage conventions" as recited in claim 22, let alone indicates them with any "indicator."

This is not found persuasive because applicant is arguing against the references separately. One cannot show nonobviousness by attacking references individually where the rejections are based on combinations of references. See *In re Keller*, 642 F.2d 413, 208 USPQ 871 (CCPA 1981); *In re Merck & Co.*, 800 F.2d 1091, 231 USPQ 375 (Fed. Cir. 1986).

The February 2005 Advisory Action further confuses the issues: it merely states that unspecified portions of one or both of two other references might be used to supplement the Goetz '913 patent, in some unspecified way, to meet the specific claim limitation at issue. The February 2005 advisory Action does not designate the portions relied on, explain their pertinence, differentiate whether the examiner proposes to modify Goetz with Brender and Murphy, or combine them as they stand, or some other combination. The Advisory Action does not discuss motivation to modify/combine, or "reasonable expectation of success:"

4. "No "Clear Issue" Is Developed for Claim 22"

Applicant is additionally in error in this statement. If applicant were to have referred to the rejection of claim 22 at paragraph 52 of the February 2004 office action, he would have recognized that the rejection of claim 22 was based upon a combination of references, specifically Goetz et al. in view of Brender et al. and Murphy et al. He would also have recognized from the rejection that although Goetz et al. contained the claimed indicators (52.4) that it was Brender et al. and Murphy et al. which provided the teaching of "data storage convention" portion of this particular claim. Therefore, by arguing that Goetz et al. alone did not teach the complete claimed aspect, applicant was indeed arguing the references individually, and as such, the response given was exactly that which was necessary to rebut applicant's argument.

2. Applicable Law

There are several minima that underlie any attempt to raise any rejection. First and foremost, the Supreme Court instructs that every written decision of every federal agency must "cogently explain why [the agency] has exercised its discretion in a given manner." *Motor Vehicle Manufacturers' Assn. of the United States Inc. v. State Farm Mutual Automobile Insurance Co.*, 463 U.S. 29, 48 (1983). All written agency decisions "must examine the relevant data and articulate a satisfactory explanation for its action including a rational connection between the facts found and the choice made." *Id.* at 43. No agency decision may "entirely fail[ ] to consider an important aspect of the problem, offer[ ] an explanation for its decision that runs counter to the evidence before the agency." *Id.* Patent Office publications reinforce the general principle, "It is not an applicant's responsibility to set out a clear and concise rejection ... setting out a rejection is the responsibility of the examiner." *Ex parte Gambogi*, 62 USPQ 1209, 1213 (Bd. Pat. App. & Interf. 2002).

The Patent Office has added several additional provisions on top of this floor, including 37 C.F.R. § 1.104(c)(2), which reads as follows:

> **§ 1.104 Nature of examination.**
>
> (2) In rejecting claims for want of novelty or for obviousness, the examiner must cite the best references at his or her command. When a reference is complex or shows or describes inventions other than that claimed by the applicant, the particular part relied on must be designated as nearly as practicable. The pertinence of each reference, if not apparent, must be clearly explained and each rejected claim specified.

MPEP § 2142 requires that an examiner "show" that the art suggests the claimed invention, not merely make a conclusory identification of a statutory section and three references. The Federal Circuit has regularly reiterated that an examiner must set forth "particularized findings" on each of the factors of *Graham v. John Deere*, 383 U.S. 1, 148 USPQ 459 (1966), including an identification of the difference between the claim and the prior art. Examiners are required to set out "findings" on their view of the prior art during examination. *In re Berg*, 320 F.3d 1310, 1315, 65 USPQ2d 2003, 2007 (Fed. Cir. 2003); *In re Epstein*, 32 F.3d 1559, 1570-71, 31 USPQ2d 1817, 1825 (Fed. Cir. 1995) (Plager, J., concurring) ("One function of the PTO's *prima*

facie case practice is to force the PTO examiners to set forth specific [rejections], which can be met by the applicant, and not just to make a general rejection.")

The rules require showings with "specificity," *In re Lee*, 277 F.3d 1338, 1343, 61 USPQ2d 1430, 1433 (Fed. Cir. 2002). Rules that avoid the need to read an examiner's mind are particularly important where, as here, no reasonable person could possibly have guessed at the positions that the examiner has held hidden until post-final Advisory Actions. For example, no one could have understood that the term "necessarily disjoint" covers "may overlap" because the word "necessarily" introduces some "logical" possibility of an exception.[5] No one could have guessed at Examiner Ellis view, that reversing the order of the bytes of a number would leave the value of the number unaffected.[6] No one could have understood the unwritten exceptions to a number of MPEP provisions that the Examiner harbors, until he stated them.[7] It's long past time to stop playing hide-and-seek, and put all the issues on the table.

3. First Error: 37 C.F.R. §§ 1.104(c)(2) and MPEP § 707.07(f) Have Been Breached, and Therefore Claim 22 is not Rejected at All

The October 2004 Office Action articulates no clear position on claim 22. Especially in view of the contradictory positions taken in the Advisory Actions, there is no way to know:

- Does the Examiner still consider Goetz '913, col. 17, lines 24-33 to be pertinent?
- Does the Examiner agree or disagree with the showing in the Response of July 2004, that Goetz '913, col. 17, lines 24-33 does not show the "indicator elements ... having a value indicating the data storage convention under which instructions from the associated regions are to be executed?"
- What other references might be pertinent to the "indicator elements" of claim 22? The June 2005 Advisory suggests that other references might be pertinent, but does not "designate the particular portions relied on."
- What modification is proposed, and what is the motivation to combine or modify?

---

[5] See discussion of claim 104 in Advisory Action of March 27, 2003.

[6] Compare Advisory Action of April 14, 2004, the first time this position is articulated, to Declaration of David R. Levine, April 10, 2003.

[7] For example, in the paper of February 10, 2003, Examiner Ellis states that he believes that "reasonable expectation of success" only applies in the chemical arts, not "all arts" as stated in MPEP § 2142.

- Would the proposed modification "render the prior art unsatisfactory for its intended purpose?"
- What is the basis for "reasonable expectation of success?"[7]

On the current state of the record, it is impossible to identify what the Examiner's position is now, let alone what the "issue for appeal" is:

- Do and Examiner agree or disagree on claim scope?
- Do and Examiner agree or disagree on the content of Goetz '913, col. 17, lines 24-33?
- Do and Examiner agree or disagree on the content of other unspecified references?
- Do and Examiner agree or disagree on the law of obviousness?
- Do and Examiner agree or disagree on the application of the law to the facts?

Until the Examiner complies with the procedural requirements of 37 C.F.R. §§ 1.104(c)(2) and MPEP § 2142-2143.03, by designating the portions of each reference relied upon, and explaining the pertinence of these references, no substantive response is possible. Until the examiner fully "answers all material traversed," MPEP § 707.07(f), it is impossible to frame an issue that is within the appellate jurisdiction of this Board. Until the Examiner complies with "applicable departmental regulations," including 37 C.F.R. § 1.104, the Office Actions are "illegal and of no effect." *Accardi*, 359 U.S. at 545; *Schroeder*, 212 F.3d at 1269-70. Claim 22 is not rejected.

Further, an appeal brief cannot rebut showings that have not been made, or identify dependent claims that should stand or fall with claim 22.

The Board should order the Examiner to either set forth a full examination of claim 22, or allow it.

4. Second Error: Incorrect Proposition of Law

Though nothing is clearly stated, it appears that the Examiner may be creating new elements that have no correlate in the prior art. It appears that he may recognize that no reference teaches or suggests "indicator elements each having a value indicating the data storage convention under which instructions from the associated region are to be executed."

Obviousness may never be founded on new components that are not taught in the prior art. MPEP § 2143.03 states that there is no obviousness unless every element of the claim is

shown to exist in the prior art. *Motorola v. Interdigital Technology Corp.*, 121 F.3d 1461, 1466-67, 43 USPQ2d 1490, 1490-91 (Fed. Cir. 1997) (reversing a jury verdict of obviousness because an element was not taught in the particular art relied upon, even though that element was known elsewhere).

5. Third Error: Incomplete and Faulty Technological Analysis

If the Examiner believes that it would be obvious to take Goetz' existing mechanisms, remove them, and replace them with entirely different functionality, then Appellant respectfully requests the three showings demanded by MPEP § 2143, including a showing that this would not "render the prior art unsuitable for its intended purpose" and would "reasonably be expected to succeed." No such showings have been made.

6. Fourth Error: On Technological Grounds, Goetz '913 is Not Combinable with Brender '422 or Murphy '947

This is discussed in full at § VI.E.4 at page 25.

7. Dependent Claims 23-36

Because the Examiner has not stated a clear position, it is impossible to identify what claims should stand or fall with claim 22.

Pursuant to the waiver of requirements for timely action, see § IV.A, claims that stand or fall with claim 22 need not be designated now, and will be designated at an appropriate point in the future.

C. Claim 51

Claim 51 recites as follows:

> 51. A method, comprising:
>
> storing instructions in pages of a computer memory managed by a virtual memory manager, the instruction data of the pages being coded for execution by, respectively, computers of two different architectures and/or under two different execution conventions;
>
> in association with pages of the memory, storing corresponding indicator elements indicating the architecture or convention in which the instructions of the pages are to be executed, the pages' indicator elements being stored in a table whose **entries** are **indexed** by **physical** page frame number;

executing instructions from the pages in a common processor, the processor designed, responsive to the page indicator elements, to execute instructions in the architecture or under the convention indicated by the indicator element corresponding to the instruction's page.

1. First Error: Procedural Failure to Answer All Material Traversed

Claim 51 is discussed at ¶ 29 of the February 2004 Office Action, and ¶ 14.2 of the October Office Action, in the context of Goetz '913 alone. The Advisory Actions of 2/14/05 and 6/7/05 are totally silent on claim 51.

Appellant notes that the following discussion is essentially a repetition of the Response to Office Action of January 2005. However, the Advisory Actions of February 2005 and June 2005 fail to "Answer All Material Traversed," in violation of MPEP § 707.07(f). Any rejection has passed away on procedural grounds. Until the Examiner states a position that reflects the ordinary understanding of the terms "index," "entries," and "physical page frame number" and the very dictionary definition he relies upon, Appellant is unable to meaningfully respond further.

There is facially no attempt to answer all material traversed in the Requests for Reconsideration, as required by MPEP § 707.07(f). Any rejection has lapsed.

The Board should order the Examiner to either set forth a full examination of claim 251, including an answer to all material traversed, or allow it.

2. Second Error: Faulty Technological Analysis

The Office Action suggests that the "table whose entries are indexed by physical page from number" corresponds to Goetz '913, Fig. 4, and col. 10, lines 4-28, which read (in pertinent part) as follows (hand-written box and annotations added):

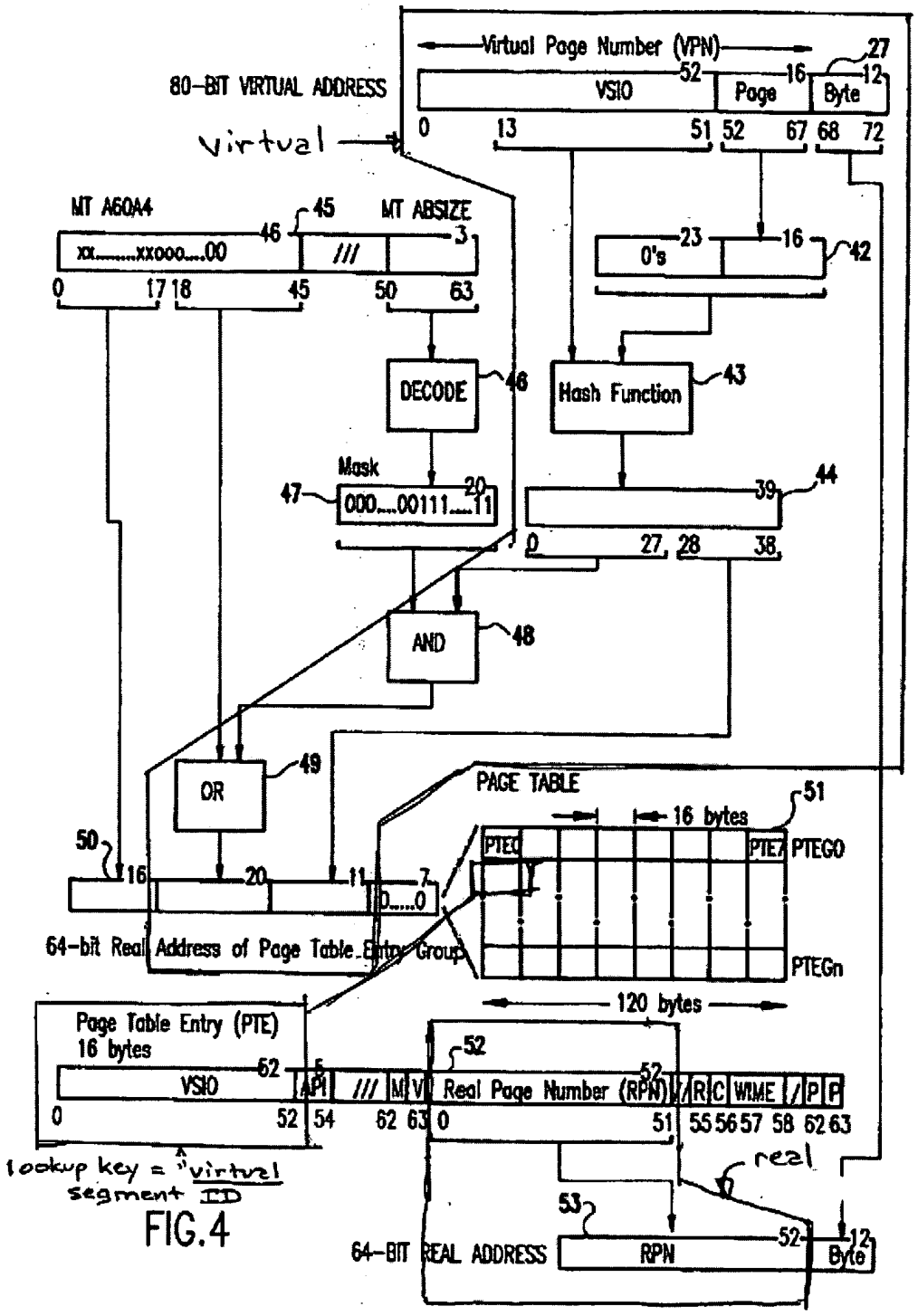


FIG.4

.... In the PowerPC architecture as shown in FIG. 4, VAs [virtual addresses] in register 27 are translated to physical addresses (PAs) via a hashed page table search. More particularly, bits 52 to 67 (page field) of the 80-bit virtual address in register 27 [are combined with other values] to form ... page table origin register 50. ... The highest seven bits of register 50 are forced to zero to form the real

> address of the page table 51 in memory. The individual page table group entries are searched until an entry 52 is found whose virtual segment ID matches that of the original VA. When found, the real page number is extracted from the page table group entry 52...

That is, the virtual address is used to determine the real address of the page table. Then the individual entries of the page table are searched based on the virtual segment ID to find a particular entry. Only after an individual page table entry is located based entirely on the virtual address of the address to be translated is a real address of the target page extracted from the entry.

This portion of Goetz '913 states four things that are incompatible with the interpretation expressed in the Office Action:

(a) Register 50 contains the real address of only the "page table" 51 as a whole. No real address is ever used to "index" an "entry," as recited in claim 51.

(b) To find a particular entry in the page table (as recited in claim 51) Goetz '913 uses various portions of the "virtual" address, not any value related to any "physical" address, as recited in claim 51.

(c) Goetz relies on the difference between "real" addresses (or "physical address," as recited in claim 51) and "virtual" addresses for correct behavior.

(d) Goetz '913 "extracts" the "real page number" from the page table entry that was located based on its "virtual segment ID."

The contrary view expressed in the Office Actions, that the "real address" of a page is used to "access a particular element" from which Goetz then "extracts" the real page number of the very page whose entry is indexed, makes no engineering sense. Using a real address to index a table to "extract" the very same real address is just silly. The scheme proposed in the October Office Action is essentially the same as looking in the white pages of the phone book to find the correct spelling of a person's last name (the real page number), after one already knows the correct spelling of the last name. No person would rationally do such a thing, or design such a step into a computer – clearly Goetz '913 does not teach the technique set out in the Office Action.

3. Third Error: Selective Quotation from Sources

The October 2004 Office Action further confuses the issues by omitting the relevant portion of the Microsoft Computer Dictionary. In pertinent part, the relevant Microsoft definition of "index" reads as follows:

> **index** ... In programming, a scalar value that allows direct access into a multielement data structure such as an array. The index allows the programmer to ... derive the location of the desired element. ...

The "elements" of the second sentence of this definition are the "entries" recited in claim 51. Because this second sentence was entirely omitted from the Office Action, the Examiner's view with respect to the "elements" of the definition, or the "entries" of claim 51, cannot be discerned.

Further, the first sentence of the Microsoft Dictionary uses the word "into," to clarify that "index" refers to access to the "elements" of the second sentence. Merely ascertaining the location of the "multielement data structure" in bulk, with no differentiation among the "elements," is not "indexing."[8]

4. Dependent Claims 52-59

Claims 52-59 are not allowed. However, because of the procedural breaches discussed above, these claims are not rejected.

Claims 52-59 stand or fall with claim 51.

D. Claim 63

Claim 63 recites as follows:

> 63. A microprocessor chip, comprising:
>
> an instruction unit, configured to fetch instructions from a memory managed by the virtual memory manager, and configured to execute instructions coded for first and second different computer architectures or coded to implement first and second different data storage conventions;
>
> the microprocessor chip being designed (a) to retrieve indicator elements stored in association with respective pages of the memory, each indicator element indicating the architecture or convention in which the instructions of the page are

---

[8] The current 5th edition of the Microsoft Computer Dictionary confirms that an "indexed search," using a physical page frame number as a search key, would be equivalent to "indexing" in the sense used in the older edition's definition, subject to the limits of the prior art. Goetz '913 raises no such issues, since anything in Goetz '913 relating to "indexing" of "entries" is based on virtual address.

to be executed, and (b) to recognize when instruction execution has flowed or transferred from a page of the first architecture or convention to a page of the second, as indicted by the respective associated indicator elements, and (c) to alter a processing mode of the instruction unit or a storage content of the memory to effect execution of instructions in accord with the indicator element associated with the page of the second architecture or convention;

wherein the indicator elements are stored in **a table distinct from a primary address translation table and from portions of the primary address translation table cached in a TLB (translation lookaside buffer)** used by a virtual memory manager, the indicator elements of the table being stored in association with respective pages of the memory.

1. Procedural History of Claim 63

Claim 63 is discussed at ¶¶ 32 and 34 of the February 2004 Office Action. However, the February 2004 Office Action only "designates" portions of Goetz '913, portions that refer to dozens of circuits and components. The February 2004 Action does not provide any clear explanation of "pertinence," as required by 37 C.F.R. § 1.104(c)(2). The table of claim 63 is apparently compared to two entirely different and unrelated objects in Goetz '913. It is impossible to tell whether the Office Action compares the "indicator elements" of claim 63 to the "P Bit," the "Q bit," or another bit.

The October 2004 Office Action, at ¶ 14.3, is the first to contain an explanation of "pertinence," and resolves some of the ambiguity.

The February 2005 and June 2005 Advisory Actions are silent with respect to any substantive issue of claim 63.

There is no dispute that amendments to claim 63 were made at the earliest possible time after the Examiner began to clarify his explanation of pertinence, and that the amendments are necessary. Both the Examiner and the T.C. Director have been totally silent on 37 C.F.R. § 1.116(b)(3). Neither has acted to deny entry of the amendment. The time for doing so is past. The amendment to claim 63 is entered.

The Board should order the Examiner to either set forth a full examination of claim 63, including an answer to all material traversed, or allow it.

2. Claim 63 is Patentable

The "indicator elements" stored in "a table distinct from a primary address translation table and from portions of the primary address translation table cached in a TLB (translation lookaside buffer)" distinguishes Goetz' TLB (the only element of Goetz '913 mentioned in any Office Action). Claim 63 is therefore patentable over Goetz '913.

E. Claim 87

Claim 87 recites as follows:

> 87. A method, comprising the steps of:
>
> executing a control-transfer instruction under a first execution mode of a computer, the instruction being architecturally defined to transfer control directly to a destination instruction for execution in a second execution mode of the computer;
>
> before executing the destination instruction, altering the data storage content of the computer to establish a program context under the second execution mode that is logically equivalent to the context of the computer as interpreted under the first execution mode, the reconfiguring including at least one data movement operation not included in the architectural definition of the control-transfer instruction.

1. Procedural History of Claim 87

Since its original filing in 1999, claim 87 has recited a "control-transfer instruction of a computer ... architecturally defined to transfer control directly to a destination instruction." Claim 87 is the only claim that recites anything analogous to this limitation.

The February 2004 Office Action (paper no. 38) makes a summary statement that claim 87 is rejected, but nowhere actually states a rejection in the manner required by 37 C.F.R. § 1.104 and MPEP § 2141-2143.03. The sum total discussion of claim 87 is as follows:

46.	**Claims 3, 15-16, 18-19, 21-33, 42-47, 49-50, 54-59, 69, 73-85, 87-93, 96-103, 110-112, 116-126, and 133 are rejected under 35 USC § 103 as being unpatentable over Goetz et al., U.S. Patent 5,854,913, in view of Brender et al., U.S. Patent 5,339,422 and Murphy et al., U.S. Patent 5,764,947 (incorporated by reference into Brender et al. at col. 1 lines 10-12 and 19-24).**

> **100. As to claims 87-93, 96-103, 110-112, 116-117, 118-126, and 133, they do not teach or define above the invention claimed in the previously rejected respective claims and are therefore rejected under Goetz et al. in view of Brender et al. and Murphy et al. for the same reasons set fourth in the previous claim rejections, supra.**

There is no dispute that the February 2004 Action does not designate "the particular part relied on" of any reference, or "explain the pertinence" of any reference, as required by 37 C.F.R. § 1.104(c)(2), or set forth any of the other required *prima facie* elements of any rejection of claim 87. There is likewise no dispute that the February 2004 Office Action never designates any reference or portion thereof as corresponding to the "control transfer instruction" of claim 87 (or any other claim, because this instruction is not recited in any other claim) to any reference.

In Appellant's paper of July 12, 2004, Appellant noted the silence of the Office Action with respect to claim 87, and that this silence represented a failure to raise any rejection whatsoever. Response of 7/12/04 at 36-37.

The October 2004 Office Action partially filled the silence of the earlier Action. The sum total of the discussion of claim 87 in October 2004 is as follows:

> **14.4. That: "Claim 87 recites "executing a control-transfer instruction ... architecturally defined to transfer control directly to a destination instruction". ... Further, Brender '422 teaches that the control transfer instruction must transfer control to the jacket, not "directly to a destination instruction" as recited in claim 87".**
>
> **This is not found persuasive because Brender et al. does indeed indicate that the control-transfer instruction is "architecturally defined to transfer control directly". See col. 15 lines 48-50: "In the present embodiment, it is based on the fact that a direct X call to a Y routine incurs an X operand fault". Accordingly, Brender et al. indicates that the call is "architecturally defined" to transfer "directly". Brender et al.'s system simply intervenes to convert between the expected data storage conventions of the two routines.**

There is no dispute that this ¶ 14.4 is the first mention in the entire prosecution history of col. 15, lines 48-50 of the Brender reference, or anything relating to the subject matter discussed there. There is likewise no dispute that neither the February 2004 nor the October 2004 Office Action has ever set out a rejection in even minimal conformance with 37 C.F.R. § 1.104(c)(2) and MPEP §§ 2141-2143.03 by "designating" the portions relied on, "clearly explaining" the

pertinence, showing motivation to modify/combine to meet claim 87, or any "reasonable expectation of success."

Appellant's paper of January 25, 2005, at page 41-42 shows that the Examiner's technological analysis of October 2004 is incorrect.

The Examiner's papers of February 14, 2004 and June 7, 2005 are absolutely silent in response. The Examiner failed to "Answer all material traversed" as required by MPEP § 707.07(f).

2. First, Second and Third Errors: Because of Procedural Omissions, No Rejection of Claim 87 Exists

No Office Action has ever addressed the language of claim 87, an "instruction" with certain properties. Instead, the Office Action points to text in Brender '422 that discusses transferring control between "routines." Until an Office Action addresses the precise language of claim 87, no rejection exists.

Claim 87 is briefly mentioned in ¶ 100 of the Office Action of February 2004, in relation to some unspecified portions of Goetz '913, Brender '422, and Murphy '947, combined in some unspecified way. No language of claim 87 is compared to any reference, and thus no rejection was raised. The October 2004 Action, ¶ 14.4, compares an incorrect paraphrase of single limitation of claim 87 to Brender '422, col. 15, lines 48-50, but makes no comparison of claim 87 as a whole to any prior art. The Advisory of 2/14/2005 ¶ 5 mentions claim 87, but does not "answer all material traversed" in the Response to Office Action of 1/25/2005, pages 40-42. Because the Examiner's papers do not observe procedural requirements, any rejection that may have existed at one time has ceased to exist.

The Director instructs, as a matter of "procedure," that the examiner bears the initial burden to come forward with three showings to support any rejection for obviousness:

> **2142 Legal Concept of *Prima Facie* Obviousness**
>
> The legal concept of *prima facie* obviousness is a procedural tool of examination which applies broadly to all arts. ... The examiner bears the initial burden of factually supporting any *prima facie* conclusion of obviousness. ...
>
> **ESTABLISHING A *PRIMA FACIE* CASE OF OBVIOUSNESS**

> To establish a *prima facie* case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all the claim limitations.

The Examiner disagrees with the Director. No office Action has ever addressed "reasonable expectation of success" for any combination of Goetz '913 with either Brender '422 or Murphy '947.[9] Examiners do not have discretion to rewrite the rules – until an Office Action addresses all elements of *prima facie* obviousness, no obviousness rejection exists.

The Board should order the Examiner to either set forth a full examination of claim 87, or allow it.

3. Fourth Error: Technologically, Brender '422 Makes Clear That No "Instruction" Meeting Claim 87 Can Exist

The October Office Action misstates the content of Brender '422. Brender '422 makes clear that no "instruction" meeting claim 87 can exist.

First, Brender '422, col. 8, lines 31-37 and col. 10, line 43 through col. 15, line 16 teaches that his compiler identifies every transfer of control between different instruction sets, and then inserts a software "jacket" at that transfer. Brender's "jacket" is exactly analogous to the pipe adapter discussed in § V, above. Brender's "jacket" is interposed between the source "instruction" and destination of the transfer – there is never a "direct" transfer at an instruction level, as recited in claim 87.

Second, Brender '422 then absolutely ensures that there is no possibility of a direct transfer from a source instruction to the target of that instruction (across an instruction set difference), except through the jacket: before "Y" routines are linked with "X" routines ("X" and "Y" being the two instruction set architectures discussed in Brender '422), "The names of the called [Y] routines are hidden at link time..." (col. 15, lines 42-44). Because the names of

---

[9] In spite of several requests for some showing of "reasonable expectation of success," the Examiner refuses to provide any such showing. Indeed, in the paper of February 10, 2003, Examiner Ellis states that he believes that "reasonable expectation of success" only applies in the chemical arts, not "all arts" as stated in MPEP § 2142.

the "Y" routines are hidden before they are linked with "X" routines, the linker cannot possibly link an "X" instruction to "transfer control directly to a destination instruction" in the "Y" instruction set. Brender '422 must do this to ensure that the software jacket is interpolated on every transfer between instruction sets.

Third, Brender '422 expressly teaches away from any "direct" transfer of control across an instruction set boundary, without going through the jacket. Such transfers are blocked as illegal. Brender '422 teaches that these instructions are architecturally defined to fault, not to "transfer control" as recited in claim 87. Paragraph 14.4 of the October Office Action quotes one of the clearest of these statements, Brender '422, col. 15, lines 48-50:

> In the present embodiment, it is based on the fact that a direct X call to a Y routine incurs an X operand fault.

This sentence from Brender '422 teaches opposite the interpretation suggested in the Office Action: an attempted "direct call" is architecturally defined to "fault," that is, to not complete at all. Such an instruction is not architecturally defined to "transfer control directly" to a destination, as recited in claim 87.

Because any correspondence between this language of claim 87 and any prior art rests on faulty technical analysis, claim 87 may be allowed.

4. Fifth Error: On Technological Grounds, Goetz '913 is Not Combinable with Brender '422 or Murphy '947

Goetz '913 uses an approach that is fundamentally incompatible with the approach of Brender '422 or Murphy '947. They address different problems in different ways, using incompatible techniques.

Goetz '913 teaches hardware that is capable of executing either a rough approximation of the Intel X86 (now Pentium) instruction set (a "complex instruction set computer," or "CISC"), and a modified IBM/Motorola PowerPC (a "reduced instruction set computer" or "RISC") instruction set (Goetz '913, Fig. 8). Goetz' overall approach is to modify one architecture or the other by dropping a feature here or adding a feature there to eliminate any data incompatibility between the two architectures. *E.g.*, Goetz '913, col. 18, lines 63-64 (X86 data format is "completely replaced" with a modified PowerPC format).

The only cases where Goetz '913 allows a difference to persist is where a switch between the two architectures can be implemented with no software intervention or change to state, beyond simply switching the instruction pipeline from X86 mode to PowerPC mode. For example, Goetz '913, col. 15, lines 26-29 discusses the difference between Intel X86 80-bit floating-point registers and PowerPC 64-bit floating point registers. No adjustment is required on transition. Goetz' CPU leaves all the data untouched: Goetz´913 expressly states that floating point data is left in "the same register files," Goetz '913, *id.*, all he has to do is enable or disable 16 bits of the floating-point data path. No modification of data storage content is required for such differences.

In contrast, Brender '422 and Murphy '947 are directed to systems for emulating an Alpha (the 1990's RISC computer developed by Digital Equipment Corp.) on a VAX (Digital's 1980's CISC processor), and vice-versa. Neither Brender '422 nor Murphy '947 mention any specialized hardware to allow one computer to execute binaries for the other.

Both Brender '422 and Murphy '947 take similar approaches: both analyze the program to be executed to identify every conceivable point at which execution crosses from VAX binary code to Alpha binary code (in some cases, after binary translation from one instruction set to the other). The overwhelming bulk of both Brender's and Murphy's disclosures are directed to the various cases that may arise: how they are detected, and how they are handled. Based on that analysis, Brender '422 and Murphy '947 both build a data structure for each and every case[10] of cross-architecture call that can be detected, Murphy '947, col. 6, lines 6-7 ("each incoming call"), col. 6, lines 14-15 ("outbound call"); Brender '422, col. 8, lines 32-34 ("each X [VAX] and Y [Alpha] routine"), col. 10, lines 50-53 ("Jacketing requires that knowledge of all the relevant calling convention characteristics for each subprogram in each domain be available to the Jacketing mechanism.").

There are three differences between Goetz '913 on the one hand, and Brender '422 and Murphy '947 on the other, that defeat any combination:

---

[10] In many cases, the data structures will have identical values, and may be collapsed together. Nonetheless, both Brender '422 and Murphy '947 start by analyzing each conceivable transfer as a particular case.

The "principle of operation" for Goetz '913 is to modify the two architectures – add some features here, remove some there – until any switch between them can be effected smoothly. In contrast, both Brender '422 and Murphy '947 must take both architectures without modification. For example, Brender '422 states that his goal is to "execute, test and debug ... software designed for a new hardware architecture" – his whole purpose would be defeated if the new architecture were modified as taught in Goetz '913. Similarly, Murphy '947 is directed to providing a migration tool for VAX software onto Alpha hardware for customer software[11] – that purpose would be defeated if the VAX were modified in the manner taught by Goetz'.

Because Goetz '913 has already "ironed out" all of the differences between the two architectures, he provides no mechanism by which software could assist in a cross-architecture switch. Goetz '913, col. 17, lines 41-44 teaches that all that is required on an architecture switch is wait a few cycles for the pipeline to drain, "without exceptions" or other control transfer that might allow software to gain control. In contrast, Brender '422 and Murphy '947 exhaustively analyze every possible case to make sure that a software jacket is invoked on every switch between architectures.

Goetz '913 is implemented essentially entirely in hardware, while Brender '422 and Murphy '947 are implemented entirely in software.

Difference no. 1 defeats "motivation to combine" – the "principle of operation" of one or the other would have to be abandoned in order to combine Goetz '913 with either Brender '422 or Murphy '947. MPEP § 2143.01 disallows an obviousness combination in such circumstances ("If the proposed modification or combination of the prior art would change the principle of operation of the prior art invention being modified, then ... the references are not sufficient to render the claims *prima facie* obvious.")

Difference no. 2 defeats "reasonable expectation of success:" Brender's or Murphy's software cannot work on Goetz' machine because Goetz expressly states that his switch occurs "without exceptions" or other mechanism that might allow software to be invoked on a switch. Goetz '913, col. 17, lines 41-44 (switch occurs "without exceptions" triggered by the switch itself).

---

[11] *See* Murphy '947 at col. 2, line 50-55; Anton Chernoff, et al., FX!32, A Profile-Directed Binary Translator, IEEE Micro, Vol. 18 No. 2 pp. 56-64 (March/April, 1998); Raymond J. Hookway, Mark A. Herdeg, Digital FX!32: Combining Emulation and Binary Translation (August 28, 1997), http://www.digital.com/info/DTJP01HM.HTM; Digital Equipment Corp., White Paper: How DIGITAL FX!32 Works (September 1997) http://www.digital.com/semiconductor/amt/fx32/fx-white.htm, all three of which are of record in this application.

Difference no. 3 also defeats "reasonable expectation of success." Goetz '913 is implemented essentially entirely in hardware. When setting up a fab line for a chip design takes several weeks and several million dollars, no reasonable person in the art would even consider implementing Brender's and Murphy's data structures, which are specialized to one single program, in hardware. That enormous cost would have to be borne for each version of each program. Further, when hardware costs rise with the cube of die size, no sane person would consider implementing Brender's and Murphy's data structures, one specifically created for each conceivable transition, in hardware.

Fourth, neither Goetz '913, Brender '422 or Murphy '947 teach any technique that would bridge these differences. One confronted with these three references (assuming for the moment that "motivation" to pick these three existed), in order to make them work together, would have to remove several crucial features of the prior art, and design substantial "glue" hardware and/or software that is taught in no reference.

Paragraph 14.6 of the October 2004 Office Action attempts to justify the combination by showing that "software and hardware are logically equivalent and interchangeable." This paragraph of the Action misstates the law, and misstates the engineering realities.

Fifth, paragraph 14.6 is no more than a statement that "references can be combined or modified," reasoning that is forbidden at MPEP § 2143.01. No Office Action has ever shown motivation to combine these references.

Sixth, Brender's and Murphy's software techniques cannot "reasonably" be implemented in hardware: both Brender '422 and Murphy '947 require frequent modification that is not practical in hardware. Brender '422 teaches that an entire program must be compiled and linked, if any single instruction changes, on the same basis as any other program. Brender '422, col. 7-16, see esp. col. 15, lines 24-57 (both "manual" and "automatic" jacketing require modification of the binary image at "image build time"); Murphy '947, col. 7, lines 6-15. Hardware cannot be generated on such a "throw-away" basis in any cost-effective manner – for example, any such attempt would require several weeks of turn-around time and (typically) hundreds of thousands of dollars. Instead, hardware is designed to be general, so that it need not be re-fabricated for each revision. Accordingly, Appellant's specification discloses a few simple, general structures

and techniques that can be adapted to allow cross-instruction-set calls in many different programs. See, e.g., sections I-IV of the specification, pages 31-75.

Seventh, a key difference between hardware and software is the cost curve. The cost of software goes up roughly linearly with capability. In contrast, the cost of hardware goes up roughly as the cube of die size, or roughly as the cube of chip capability. See lecture slides from the University of California at Berkeley, available at vco.ett.utu.fi/courses/ETT_2015/kalvot/luento4.pdf, excerpts attached hereto as the Exhibit in the Evidence Appendix. Therefore, most CPU chip designs avoid large on-chip structures that are not directly related to the core tasks of executing instructions. Because Brender '422 and Murphy '947 are entirely software based, they can use large structures that were not amenable to hardware implementation at the time of the invention. For a first example, Brender '422 and Murphy '947 teach that an entire program is linked together for execution. Brender '422, col. 15, lines 24-57; Murphy '947, col. 7, lines 6-40. Implementing an entire program in hardware would not be recognized as desirable by one of ordinary skill. For a second example, Murphy's and Brender's "jacketing tables" and "jacketing routines" may be quite large, because a specific data structure must be generated for each routine that could conceivably be called from the opposite instruction set. E.g., Murphy '947, col. 6, lines 4-8 ("an array that forms a signature ... for each incoming call in each Y code routine..."); Brender '422, col. 8, lines 31-37 ("Jacketing requires that knowledge ... for each subprogram in each domain..."); Brender '422, col. 10, line 43 through col. 15, line 16. Because Brender '422 and Murphy '947 use entirely-software-based approaches, they have no motivation to teach techniques that could practically be implemented in hardware. Implementing their large, non-general structures in hardware would be neither reasonably practical ("reasonable expectation of success") nor desirable ("motivation to combine") to one of ordinary skill.

All rejections based on combinations of Goetz '913 with either Brender '422 or Murphy '947 may be withdrawn.

5. Dependent claims 88-93 and 134 and 135

These claims stand or fall with claim 87.

F. Claim 94

Claim 94 is discussed at paragraphs 43-44 of the Office Action of February 2004, and in paragraph 14.5 of the October 2004 Action, in the context of Brender '422 alone (with Murphy '947 incorporated by reference). Claim 94 recites as follows:

> 94. A method, comprising the steps of:
>
> executing a section of computer object code twice, without modification of the code section between the two executions, the code section materializing a destination address into a register and being architecturally defined to directly transfer control indirectly through the register to the destination address, the two executions materializing two different destination addresses;
>
> the two destination code sections at the two materialized destination addresses being coded in two distinct instruction sets and, respectively, obeying the default calling conventions native to each of the two instruction sets, neither instruction set being a subset of the others.

1. First Error: Procedurally, the Office Actions are Too Incomplete to Raise Any Rejection

Even taken together, the discussion in the two Office Actions is too incomplete to constitute a rejection. For example, claim 94 recites a "register" that performs certain functions. The Office Actions do not address this word of claim 94, and fail to indicate any "register" that is thought to perform these functions. No rejection exists.

The February 2005 Advisory Action makes no response to the substantive arguments on claim 94 made in Appellant's papers. For failure to "answer all material traversed," any rejection has lapsed.

The Board should order the Examiner to either set forth a full examination of claim 94, or allow it.

2. Second Error: Faulty Technological Analysis

As noted in § VI.E.3 at page 24 above, the Examiner's technological analysis of Brender '422, col. 21, lines 50-57 is incorrect. In fact, the paragraph following the second paragraph cited in the Office Action, Brender '422 does discuss a "register," and shows that it cannot possibly meet the "register" of claim 94:

> The only change is to replace the load of R26 from the second quadword of the target procedure descriptor with a load of the code address for [the runtime

routine that implements cross-ISA calls]. The code address for the [runtime routine] is loaded as opposed to a procedure value address; [the runtime routine] is invoked with a non-standard call and there is no associated procedure descriptor.

Brender '422 makes clear that the "destination address" materialized into register R26 is "always" the address of the special jacketing runtime routine, not a "direct" destination.

Claim 94 is neither rejected nor rejectable over Brender '422.

G. Claim 96

Claim 96 recites as follows:

96. A microprocessor chip, comprising:

two instruction decoders designed to decode instructions of first and second instruction sets, respectively, and circuitry of a single instruction pipeline designed to execute the instructions decoded by either of the two instruction decoders;

circuitry and/or software designed to detect when execution flows or transfers control from code coded in one instruction set to code coded in the other, program code in the first and second instruction sets using first and second different data storage conventions, respectively; and

circuitry and/or software designed to respond to the detection by altering the data storage content of the computer to create a program context under the second data storage convention that is logically equivalent to a pre-alteration program context under the first data storage convention.

1. First Error: The Office Actions Are Too Incomplete to Raise Any Rejection

Claim 96 is briefly mentioned, without being rejected, in paragraph 100 of the Office Action of February 2004, over unspecified portions of Goetz '913, Brender '422, and Murphy '947, combined in some unspecified way. A single limitation of claim 96 is discussed in paragraph 14.4 of the October 2004 Action. The claim as a whole has never been compared element-by-element to the art at issue, the "portions of the references" relied on has never been designated, and no expalantion of "pertinence" has ever been provided. No rejection exists.

The Advisory Action of February 2005 is silent the substantive arguments on claim 96 made in Appellant's papers. For failure to "Answer All Material Traversed," any rejection has lapsed procedurally.

The Board should order the Examiner to either set forth a full examination of claim 96, including an answer to all material traversed, or allow it.

2. Second Error: the References are Not Combinable

Claim 96 is patentable because Goetz '913 is not combinable with Brender '422 and Murphy '947, as discussed in § VI.E.4 at page 25, above.

H. Claim 104

Claim 104 recites as follows:

> 104. A method, comprising the steps of:
>
> executing instructions fetched from first, second and third regions of a single address space of the memory of a computer, the instructions of the first region being coded for execution by computers of a first architecture and following a first data storage convention, **the instructions of the second region being coded for execution by computers of a second architecture and following the first data storage convention**, the instructions of the third region being coded for execution by computers of the second architecture and following a second data storage convention, the memory regions having associated modifiable indicator elements, a hardware structure for storing the indicator elements enforcing a requirement that the memory regions be necessarily disjoint, the modifiable indicator elements each having a value indicating the architecture and data storage convention under which instructions from the associated region are to be executed;
>
> when execution of the instruction data flows or transfers between the first, second and third regions, adapting the computer for execution in the architecture and/or convention of the region transferred to.
>
> when execution of the instruction data flows or transfers between the first, second and third regions, adapting the computer for execution in the architecture and/or convention of the region transferred to.

1. Procedural History of Claim 104

The February 2004 Action discusses claim 104 under § 102 over a single reference:

> 10. Claims 1-2, 4-14, 17, 37-47, 51-53, 61-66, 68, 70-72, 104-109, 113-115, and 128-132 are rejected under 35 USC § 102(e) as being clearly anticipated by Goetz et al., U.S. Patent 5,854,913.

42.	**As to claim 104-109, 113-115, 128-132, they do not teach or define above the invention claimed in the previously respective rejected claims and are therefore rejected under Goetz et al. for the same reasons set fourth in the previous claim rejections, supra.**

Appellant's July 2004 paper notes two things: (a) there is no element-by-element comparison of claim 104 to any reference, and claim 104 recites a limitation ("first and second data storage conventions") not present in other claims. Without some showing that the reference meets this limitation, Appellant is unable to respond precisely. (b) Second, and as a separate issue, the "first and second data storage conventions" language distinguishes the portions of the Goetz reference designated by the Office Action.

The October 2004 Action reads as follows:

> 14.7. That: "Claim 104 recites two memory regions that "[follow] first and second data storage conventions." As discussed above in connection with claim 22, at page 34, there is no indication that Goetz '913 uses two different "data storage conventions."
>
> This is not found persuasive because is arguing against the references separately. One cannot show nonobviousness by attacking references individually where the rejections are based on combinations of references. *See In re Keller*, 642 F.2d 413, 208 USPQ 871 (CCPA 1981); *In re Merck & Co.*, 800 F.2d 1091, 231 USPQ 375 (Fed. Cir. 1986).

Under any definition of "new ground of rejection," even the flawed "same references, same statute" test, this shift from a single reference, Goetz '913, to a "combination of references" is a new ground of rejection.

In the Advisory of 2/14/05, Examiner Ellis again shifts ground:

3.	**"Claim 104 has Not Been Examined"** **Applicant is in error in this statement. Claim 104 was examined, and was rejected, in paper number 38, mailed February 11, 2004, at paragraph 42. Claim 104 is rejected with Goetz et al. in the exact same manner as claim 4 (as claim 4 existed on February 11, 2004, due to the fact that claim 104 is simply a restatement of claim 4 (as claim 4 existed on February 11, 2004). A fact that applicant should have been immediately aware of because applicant himself wrote the claims. Accordingly, claim 104 was rejected.** **Applicant is additionally confused because paragraph 14.7 of the Office Action of October 2004 contains a copy of the response from paragraph 14.1 regarding claim 22. This**

was a typographical mistake in the office action of October 2004. The correct response was: Goetz et al. taught using "the instructions ... following first and second data storage conventions" at col. 15 lines 45-49.

There is no dispute that the Examiner has raised "new grounds" and has not been "clear" or "complete." Final rejection is premature, and the amendments are entered.

2. First Error: The Office Actions are Too Incomplete to Raise Any Rejection

Claim 104 was briefly mentioned in paragraphs 10 and 42 of the Office Action of February 2004 in the context of Goetz '913 alone. Because the Office Action of February 2004 made no attempt to set out the "designation" of portions relied on, and no clear explanation of "pertinence," of claim 104 to Goetz '913 or any other reference, no rejection was raised in February 2004.

Even the 2/14/2005 Advisory is not helpful. Claim 104 recites "two different instruction sets" and "two different data storage conventions." Goetz '913 at col. 15, lines 45-49, could be fairly construed to cover one or the other. But because the Office Actions have been silent on so many issues, it is impossible to determine how the Examienr considers that both limitations of claim 104 might correspond to the references, whether his current view is that one or all three might be involved.

The Advisory Action, for the first time, "designates" Goetz, col. 15, lines 45-49. This portion reads as follows:

> For example, X86 instructions may contain anywhere from one to fifteen bytes and be aligned on any byte boundary whereas PowerPC instructions are always four bytes in length and are therefore always aligned on a 4-byte boundary.

37 C.F.R. § 1.104(c)(2) states two separate requirements. Merely "designating" portions of a reference is only half the job, as this example demonstrates: without an explanation of "pertinence," the Examiner's view cannot be discerned. This portion of Goetz merely states that some instructions are in X86 format, and other instructions are in PowerPC format. There is no discussion of "modification" or "adapting" of "data storage content" implied in this sentence of Goetz '913.

The Board should order the Examiner to either set forth a full examination of claim 104, or allow it.

3. Any Rejection Fails on the Merits

Claim 104 recites three memory regions – a first and third in which the instruction architecture and data storage convention are matched to each other, and a second in which the instruction architecture and data storage convention are mismatched to each other.

Goetz '913 is careful to always maintain the instruction set and data storage convention in a matched pair. Goetz '913 discloses nothing analogous the second region of claim 104. Accordingly, claim 104 is not anticipated by Goetz '913.

I. Claim 136

Claim 136 recites as follows:

> 136. A method, comprising the steps of:
>
> executing instructions fetched from first and second regions of a single address space of the memory of a computer, the instructions of the first and second regions being coded for execution by computers of first and second instruction sets and following first and second data storage conventions, **the data storage conventions being established by software convention without being defined into the hardware definition of the first and second instruction sets,** the first and second regions having respective associated modifiable indicator elements, the modifiable indicator elements each having a value indicating the instruction set under which instructions from the associated region are to be executed;
>
> when the computer recognizes, based on the indicator elements, that execution has transferred, by flow or by a control transfer instruction, between the first and second regions, **adapting at least a part of the data storage content** of the computer from the data storage convention for the region transferred from to the data storage convention for the region transferred to before commencing execution in the transferred-to region, to create a program context under the data storage convention of the transferred-to region that is logically equivalent to a pre-alteration program context under the transferred-from data storage convention.

1. Procedural History of Claim 136

Claim 136 was added at the earliest point in prosecution after the Examiner first explained his position on claim 104, in the Advisory Action of 2/14/2005. Claim 136 is necessary to respond to the new designation of a new part of a reference in the Advisory Action.

Neither the Examiner nor the T.C. Director have acted to deny entry of claim 136 under 37 C.F.R. § 1.116(b)(3), the basis sought in the Amendment of 4/25/2005. The fee for addition of claim 136 was charged to Deposit Account. Claim 136 is entered.

2. Claim 136 is Patentable on the Merits

No Office Action has ever pointed to a difference in "data storage convention" that is overcome by "adapting at least a part of the data storage content." Because the Office Actions have discussed the claims in disconnected bites, the Examiner's view cannot be discerned.

Procedurally, the Office Action (even as supplemented with the late Advisory of 2/2005) is inadequate to raise any rejection. Substantively, the Office Action is wrong.

VII. Conclusion

The Amendment of 4/25/2005 is entered by operation of law.

No claim is rejected. There is nothing for the Board to affirm. Any rejection is necessarily a new ground. The Board should order the Examienr to make *prima facie* findings on each limitation of each independent claim, including "designation" portions of references relied on, and clearly explaining pertinence. The Board should order the Exameinr to make findings on all elements of *prima facie* obviousness. The Exameinr may not "sit mum."

Any rejection that may exist should be reversed.

Appellant requests that the application be passed to issue in due course. The Examiner is urged to telephone Appellant's undersigned counsel at the number noted below if it will advance the prosecution of this application, or with any suggestion to resolve any condition that would impede allowance. In the event that any extension of time is required, petitions for that extension of time required to make this response timely. Kindly charge any additional fee, or credit any surplus, to Deposit Account No. 23-2405, Order No. 114596-03-4000.

Respectfully submitted,

WILLKIE FARR & GALLAGHER LLP

Dated: November 28, 2005

By: _____

David E. Boundy
Registration No. 36,461

WILLKIE FARR & GALLAGHER LLP
787 Seventh Ave.
New York, New York 10019
(212) 728-8000
(212) 728-8111 Fax

Claims Appendix to

Response to Office Action or Appeal Brief

CLAIMS

1. (last amended 7/12/2004, allowed) A computer, comprising:

a processor pipeline designed to alternately execute instructions coded for first and second different computer architectures or coded to implement first and second different processing conventions;

a memory for storing instructions for execution by the processor pipeline, the memory being divided into pages for management by a virtual memory manager, a single address space of the memory having first and second pages;

a memory unit designed to fetch instructions from the memory for execution by the pipeline, and to fetch stored indicator elements associated with respective memory pages of the single address space from which the instructions are to be fetched, each indicator element designed to store an indication of which of two different computer architectures and/or execution conventions under which instruction data of the associated page are to be executed by the processor pipeline, the first architecture having a pre-defined, established definition, the computer providing a faithful implementation of the first architecture;

the memory unit and/or processor pipeline further designed to recognize an execution flow from the first page, whose associated indicator element indicates the first architecture or execution convention, to the second page, whose associated indicator element indicates the first architecture or execution convention, and in response to the recognizing, to adapt a processing mode of the processor pipeline or a storage content of the memory to effect execution of instructions in the architecture and/or under the convention indicated by the indicator element corresponding to the instruction's page.

2. (original) The computer of claim 1:

wherein the two architectures are two instruction set architectures;

and wherein the adapting includes controlling instruction execution hardware of the computer to interpret the instructions according to the two instruction set architectures according to the indicator elements.

3. (original) The computer of claim 1, wherein the two conventions are first and second calling conventions, and further comprising:

hardware and/or software designed to recognize when program execution has flowed or transferred from a region whose indicator element indicates the first calling convention to a region whose indicator element indicates the second calling convention, and in response to the recognition, to alter the data storage content of the computer to create a program context under the second calling convention that is logically equivalent to a pre-alteration program context under the first calling convention.

4. (last amended 7/12/2004, allowed) A method, comprising the steps of:

executing instructions fetched from first and second regions of a memory of a computer, the instructions of the first and second regions being coded for execution by computers of first and second architectures or following first and second data storage conventions, respectively, the memory regions having associated first and second indicator elements, the indicator elements each having a value indicating the architecture or data storage convention under which instructions from the associated region are to be executed, the first architecture having a pre-defined, established definition, the computer providing a faithful implementation of the first architecture;

when execution of the instruction data flows or transfers from the first region to the second, adapting the computer for execution in the second architecture or convention.

5. (original) The method of claim 4, wherein:

the regions are pages managed by a virtual memory manager.

6. (original) The method of claim 5, wherein the indicator elements are stored in a table of indicator elements distinct from a primary address translation table used by the virtual memory manager, the indicator elements of the table associated with corresponding pages of the memory.

7. (original) The method of claim 5, wherein the indicator elements are stored in a table, each indicator element associated with a corresponding physical page frame.

8. (original) The method of claim 5, wherein the entries are entries of a translation look-aside buffer.

9. (original) The method of claim 4, wherein the regions are lines of an instruction cache.

10. (original) The method of claim 4:

wherein the two architectures are two instruction set architectures;

and wherein the adapting step includes controlling instruction execution hardware of the computer to interpret the instructions according to the two instruction set architectures according to the indicator elements.

11. (original) The method of claim 10, wherein:

the regions are pages managed by a virtual memory manager.

12. (previously presented) The method of claim 11, wherein the indicator elements are stored in a table whose entries are indexed by physical page frame number.

13. (original) The method of claim 11, wherein the entry is one entry of a translation look-aside buffer.

14. (original) The method of claim 10, wherein a mode of execution of the instructions is changed without software intervention when execution flows or transfers from the first region to the second.

15. (original) The method of claim 10, wherein execution of the computer takes an exception when execution flows or transfers from the first region to the second.

16. (original) The method of claim 15, wherein the mode of execution of the instructions is explicitly controlled by an exception handler.

17. (original) The method of claim 10, wherein:

one of the regions stores an off-the-shelf operating system binary coded in an instruction set non-native to the computer, the non-native instruction set providing access to a reduced subset of the resources of the computer.

18. (original) The method of claim 10, wherein the two conventions are first and second data storage conventions, and further comprising the step of:

recognizing when program execution has flowed or transferred from a region whose indicator element indicates the first data storage convention to a region whose indicator element indicates the second data storage convention, and in response to the recognition, altering the data storage content of the computer to create a program context under the second data storage convention that is logically equivalent to a pre-alteration program context under the first data storage convention.

19. (original) The method of claim 18, wherein:

the regions are pages managed by a virtual memory manager;

one of the two data storage conventions is a register-based calling convention, and the other data storage convention is a memory stack-based calling convention.

20. (original) The method of claim 18, further comprising the steps of:

classifying control-flow instructions of a computer instruction set into a plurality of classes; and

during execution of a program on a computer, as part of the execution of instructions of the instruction set, updating a record of the class of the classified control-flow instruction most recently executed;

the adjusting process being determined, at least in part, by the instruction class record.

21. (previously presented) The method of claim 18, wherein:

the instruction data coded for execution by a first of the two instruction set architectures observes a data storage convention associated with the first architecture, and instruction data coded for execution by a second of the two instruction set architectures observes a second, different, data storage convention associated with the second architecture, a single indicator element indicating both the instruction set architecture and the data storage convention;

and further comprising the step of recognizing when program execution flows or transfers from a region using the first instruction set architecture to a region using the second instruction set architecture, and in response to the recognition, adjusting the data storage content of the computer from the first storage convention to the second.

22. (previously presented) A method, comprising the steps of:

executing instructions fetched from first and second regions of a memory of a computer, the instructions of the first and second regions being coded for execution by computers following first and second data storage conventions, the memory regions having associated first and second indicator elements, the indicator elements each having a value

indicating the data storage convention under which instructions from the associated region are to be executed;

recognizing when program execution has flowed or transferred from a region whose indicator element indicates the first data storage convention to a region whose indictor element indicates the second data storage convention, and in response to the recognition, altering the data storage content of the computer to create a program context under the second data storage convention that is logically equivalent to a pre-alteration program context under the first data storage convention.

23. (previously presented) The method of claim 22, further comprising the step of:

overlaying the logical resources of the first and second instruction set architectures onto the physical resources of the computer according to a mapping that assigns corresponding resources of the two architectures to a common physical resource of a computer when the resources serve analogous functions in the calling conventions of the two architectures.

24. (previously presented) The method of claim 22, wherein the adjusting step further comprises:

altering a bit representation of a datum from a first representation in the first convention to a second representation in the second convention, the alteration of representation being chosen to preserve the meaning of the datum across the change in data storage convention.

25. (original) The method of claim 22, wherein the adjusting step further comprises:

copying a datum from a first location to a second location, the first location having a use under the first data storage convention analogous to the use of the second location under the second data storage convention.

26. (original) The method of claim 22, wherein the adjusting step further comprises:

copying a datum from a third location to a fourth, the third location having a use under the first data storage convention analogous to the use of the third location under the first data storage convention and to the fourth location under the second data storage convention, a program for the copying being programmed to assume that exactly one of the first and third locations is no longer required by the execution of the program.

27. (previously presented) The method of claim 22, wherein

a rule for copying data from the first location to the second is determined by examining a descriptor associated with the instruction before the recognized execution flow or transfer.

28. (original) The method of claim 22, wherein the two conventions are two calling conventions.

29. (original) The method of claim 28, wherein:

one of the two calling conventions is a register-based calling convention, and the other calling convention is a memory stack-based calling convention.

30. (original) The method of claim 28, wherein:

the regions are pages managed by a virtual memory manager.

31. (previously presented) The method of claim 30, wherein the indicator elements are stored in a table whose entries are indexed by physical page frame number.

32. (original) The method of claim 31, wherein the entry is one entry of a translation look-aside buffer.

33. (previously presented) The method of claim 28, further comprising the step of:

taking a processor exception in response to the recognition, a handler for the exception programmed to copy a datum from a first location to a second location, the first location having a use under the first data storage convention analogous to the use of the second location under the second data storage convention.

34. (previously presented) The method of claim 22, further comprising the step of:

classifying control-flow instructions of a computer instruction set into a plurality of classes; and

during execution of a program on a computer, as part of the execution of instructions of the instruction set, updating a record of the class of the classified control-flow instruction most recently executed;

the adjusting process being determined, at least in part, by the instruction class record.

35. (original) The method of claim 34, wherein:

one of the two data storage conventions is a register-based calling convention, and the other data storage convention is a memory stack-based calling convention.

36. (original) The method of claim 34, wherein:

in some of the control-flow instructions, the classification is statically determined by the opcode of the instructions; and

in other of the control-flow instructions, the classification is dynamically determined based on a full/empty status of a register.

37. (last amended 7/12/2005) A computer processor, comprising:

a processor pipeline configured to alternately execute instructions of computers of two different architectures or processing conventions; and

a memory unit designed to fetch instructions from a computer memory for execution by the pipeline, and to fetch stored indicator elements associated with respective memory regions of a single address space from which the instructions are to be fetched, each indicator element designed to store an indication of the architecture or execution convention under which the instruction data of the associated region are to be executed by the processor pipeline, the first architecture having a pre-defined, established definition, the computer providing a faithful implementation of the first architecture;

the memory unit and/or processor pipeline further designed to recognize an execution flow or transfer from a region whose indicator element indicates one architecture or execution convention to another.

38. (original) The computer processor of claim 37, wherein the indicator elements are stored in a table of indicator elements distinct from a primary address translation table used by the virtual memory manager, the indicator elements of the table associated with corresponding pages of the memory.

39. (previously presented) The computer processor of claim 37, further comprising:

a translation look-aside buffer (TLB); and

TLB control circuitry designed to load the indicator elements into the TLB from a table stored in memory, the entries of the table being indexed by associated with corresponding physical page frame number.

40. (original) The computer processor of claim 37, wherein the two architectures are two instruction set architectures, and further comprising:

processor pipeline control circuitry designed to control the processor pipeline to effect interpretation of the instructions under the two instruction set architectures alternately, according to the associated indicator elements.

41. (original) The computer processor of claim 40, further comprising:

software programmed to manage a transition between the execution of a program executing in the first instruction set architecture, being an instruction set architecture native to the computer processor, and execution of an off-the-shelf operating system coded in the second instruction set, being an instruction set non-native to the computer, providing access to a reduced subset of the resources of the computer.

42. (original) The computer processor of claim 40:

each indicator element being further designed to store an indication of a calling convention under which the instruction data of the associated region are coded for execution by the processor pipeline;

and further comprising software programmed to alter the data storage content of a computer using the computer processor to create a program context under the second calling convention that is logically equivalent to a pre-alteration program context under the first calling convention;

the memory unit further designed to recognize when program execution has flowed or transferred from a region whose indicator element indicates the first calling convention to a region whose indicator element indicates the second calling convention, and in response to the recognition, to invoke the transition management software.

43. (original) The computer processor of claim 42, wherein the memory unit is designed to recognize a single indicator element to indicate both the instruction set architecture and calling convention of a region.

44. (original) The computer processor of claim 42, wherein the memory unit and software are designed to effect a transition between instruction boundaries, between execution in a region coded in the first instruction set using the first calling convention to execution in a region coded in the second instruction set using the second calling convention,

so that code at the source of the flow or transfer may effect the execution transition without being specially coded for code at the destination.

45. (original) The computer processor of claim 42, wherein:

one of the two calling conventions is a register-based calling convention, and the other calling convention is a memory stack-based calling convention.

46. (original) The computer processor of claim 42, wherein the logical resources for support of the first and second instruction set architectures are overlaid on the physical resources of the computer processor according to a mapping that assigns corresponding resources of the two architectures to a common physical resource when the resources serve analogous functions in the calling conventions of the two architectures.

47. (previously presented) The computer processor of claim 42, wherein

a rule for altering the data storage content from the first calling convention to the second is determined by examining a descriptor associated with the location of execution before the recognized execution flow or transfer.

48. (original) The computer processor of claim 42, wherein control-flow instructions of the instruction set are classified into a plurality of classes; and

the processor pipeline updates a record of the class of the classified control-flow instruction most recently executed;

the storage alteration process being determined, at least in part, by the instruction class record.

49. (original) The computer processor of claim 40, further comprising:

a transition manager designed to effect a transition between the execution of code coded in instructions of a first instruction set architecture and code coded in instructions of a

second instruction set architecture, the transition manager designed to alter a bit representation of a datum from a first representation under the first architecture to a second representation under the second architecture, the alteration of representation being chosen to preserve the meaning of the datum across the change in execution architecture.

50. (original) The computer processor of claim 40, further comprising circuitry designed to raise an exception when the computer recognizes that execution has flowed or transferred from a region whose indicator element indicates one architecture or execution convention to another.

51. (last amended 7/12/2004) A method, comprising:

storing instructions in pages of a computer memory managed by a virtual memory manager, the instruction data of the pages being coded for execution by, respectively, computers of two different architectures and/or under two different execution conventions;

in association with pages of the memory, storing corresponding indicator elements indicating the architecture or convention in which the instructions of the pages are to be executed, the pages' indicator elements being stored in a table whose entries are indexed by physical page frame number;

executing instructions from the pages in a common processor, the processor designed, responsive to the page indicator elements, to execute instructions in the architecture or under the convention indicated by the indicator element corresponding to the instruction's page.

52. (previously presented) The method of claim 51, wherein the pages' indicator elements cached in a translation look-aside buffer.

53. (previously presented) The method of claim 51, wherein the two architectures are two instruction set architectures, and further comprising the step of:

controlling the instruction execution hardware of the computer to interpret the instructions according to the two instruction set architectures according to the indicator element.

54. (previously presented) The method of claim 51, wherein the two conventions are first and second data storage conventions, and further comprising the step of:

recognizing when program execution has flowed or transferred from a region whose indicator element indicates the first data storage convention to a region whose indicator element indicates the second data storage convention, and in response to the recognition, altering the data storage content of the computer to create a program context under the second data storage convention that is logically equivalent to a pre-alteration program context under the first data storage convention.

55. (previously presented) The method of claim 54, further comprising the steps of:

storing instruction data in a third page, the instruction data of the third page being coded for execution by one of the two architectures, and observing a data storage convention associated with the other of the two architectures;

storing indicator elements indicating the data storage convention observed by the instructions of the respective pages; and

recognizing each transition of program execution from a page using the first data storage convention to a page using the second data storage convention, and in response to the recognition, adjusting the data storage content of the computer from the first storage convention to the second, and vice-versa.

56. (previously presented) The method of claim 53, wherein:

the instruction data coded for execution by a first of the two instruction set architectures observes a data storage convention associated with the first architecture, and instruction data coded for execution by a second of the two instruction set architectures

observes a second, different, data storage convention associated with the second architecture, a single indicator element indicating both the instruction set architecture and the data storage convention of the associated page;

and further comprising the step of recognizing when program execution flows or transfers from a page using the first instruction set architecture to a page using the second instruction set architecture, and in response to the recognition, adjusting the data storage content of the computer from the first storage convention to the second.

57. (previously presented) The method of claim 54, wherein the two conventions are a register-based calling convention and a memory stack-based calling convention, and further comprising the step of:

recognizing when program execution has flowed or transferred from a page using the register-based calling convention to a page using the memory stack-based convention, and in response to the recognition, adjusting the data storage content of the computer from the first calling convention to the second.

58. (original) The method of claim 54, wherein the adjusting step further comprises:

copying a datum from a third location to a fourth, the third location having a use under the first data storage convention analogous to the use of the third location under the first data storage convention and to the fourth location under the second data storage convention, a program for the copying being programmed to assume that exactly one of the first and third locations is no longer required by the execution of the program, a bit representation of the datum copied to the second location differing from a bit representation of the datum copied from the first location, the alteration of representation being chosen to preserve the meaning of the datum across the change in data storage convention.

59. (previously presented) The method of claim 54, wherein

a rule for copying data from the first location to the second is determined by examining a descriptor associated with the instruction before the recognized execution flow or transfer.

60. (original) The method of claim 54, further comprising:

classifying control-flow instructions of a computer instruction set into a plurality of classes; and

during execution of a program on a computer, as part of the execution of instructions of the instruction set, updating a record of the class of the classified control-flow instruction most recently executed;

the altering process being determined, at least in part, by the instruction class record.

61. (cancelled)

62. (cancelled)

63. (last amended 4/25/05) A microprocessor chip, comprising:

an instruction unit, configured to fetch instructions from a memory managed by the virtual memory manager, and configured to execute instructions coded for first and second different computer architectures or coded to implement first and second different data storage conventions;

the microprocessor chip being designed (a) to retrieve indicator elements stored in association with respective pages of the memory, each indicator element indicating the architecture or convention in which the instructions of the page are to be executed, and (b) to recognize when instruction execution has flowed or transferred from a page of the first architecture or convention to a page of the second, as indicted by the respective associated indicator elements, and (c) to alter a processing mode of the instruction unit or a storage

content of the memory to effect execution of instructions in accord with the indicator element associated with the page of the second architecture or convention;

wherein the indicator elements are stored in a table distinct from a primary address translation table and from portions of the primary address translation table cached in a TLB (translation lookaside buffer) used by a virtual memory manager, the indicator elements of the table being stored in association with respective pages of the memory.

64. (last amended 7/12/2004) The microprocessor chip of claim 63, wherein the indicator elements are stored in association with respective physical page frames.

65. (last amended 7/12/2004) The microprocessor chip of claim 63, wherein the indicator elements are stored in association with respective virtual pages.

66. (last amended 7/12/2004) The microprocessor chip of claim 63, wherein the indicator elements are stored in entries of a translation look-aside buffer.

67. (last amended 7/12/2004) The microprocessor chip of claim 63, wherein the indicator elements are stored in an instruction cache.

68. (last amended 7/12/2004) The microprocessor chip of claim 63, wherein a mode of execution of the instructions is changed without software intervention when execution flows or transfers from the first region to the second.

69. (last amended 7/12/2004) The microprocessor chip of claim 63, the microprocessor chip being designed to raise an exception when execution flows or transfers from the first region to the second;

and further comprising exception handler software programmed to handle the exception by explicitly controlling a mode of execution of the instructions.

70. (last amended 7/12/2004) The microprocessor chip of claim 63, wherein the architecture or convention indicator elements are stored in a table whose entries are indexed by physical page frame number, and cached in a translation look-aside buffer.

71. (last amended 7/12/2004) The microprocessor chip of claim 63, wherein the two architectures are two instruction set architectures, and the microprocessor chip controls the instruction unit to interpret the instructions according to the two instruction set architectures according to the indicator element corresponding to the pages from which the instructions are fetched.

72. (original) The microprocessor chip of claim 71, further comprising:

software programmed to manage a transition between the execution of a program executing in the first instruction set architecture, being an instruction set architecture native to the computer processor, and execution of an off-the-shelf operating system coded in the second instruction set, being an instruction set non-native to the computer providing access to a reduced subset of the resources of the computer.

73 (previously presented) The microprocessor chip of claim 71:

each indicator element being further designed to store an indication of a data storage convention under which the instruction data of the associated page are coded for execution by the instruction unit;

and further comprising software programmed to alter the data storage content of a computer using the microprocessor chip, to create a program context under the second data storage convention that is logically equivalent to a pre-alteration program context under the first data storage convention;

the microprocessor chip further designed to recognize when program execution has flowed or transferred from a region whose indicator element indicates the first data storage

convention to a region whose indicator element indicates the second data storage convention, and in response to the recognition, to invoke the transition management software.

74. (original) The microprocessor chip of claim 73, being further designed to:

to retrieve instruction data from a third page, the instruction data of the third page being coded for execution by a first of the two architectures, and observing a data storage convention of the second of the two architectures;

to retrieve indicator elements indicating the data storage convention observed by the instructions of the respective pages; and

to recognize each transition of program execution from a page using the first data storage convention to a page using the second data storage convention, and in response to the recognition, to adjust the data storage content of the computer from the first data storage convention to the second data storage convention, and vice-versa.

75. (original) The microprocessor chip of claim 73, further designed to recognize a single indicator element to indicate both the instruction set architecture and calling convention of a page.

76. (previously presented) The microprocessor chip of claim 71, further comprising hardware and/or software designed:

(a) to retrieve calling convention indicator elements stored in association with respective pages of the memory, each calling convention indicator element indicating which of a register-based calling convention or a memory stack-based calling convention is observed by instructions of the page;

(b) to recognize when instruction execution has flowed or transferred from a page of a memory-based convention to a page of the register-based calling convention, as indicated by the calling convention indicator elements associated with the respective pages, and

(c) in response to the recognition, to alter a storage content of the computer to create a program context under the register-based convention logically equivalent to a pre-alteration program context under the memory-based convention.

77. (last amended 7/12/2004) The microprocessor chip of claim 63:

wherein the two conventions are first and second data storage conventions;

and further comprising software programmed to alter the data storage content of a computer using the microprocessor chip, to create a program context under the second data storage convention that is logically equivalent to a pre-alteration program context under the first data storage convention;

the microprocessor chip being further designed to recognize when program execution has flowed or transferred from a region whose indicator element indicates the first data storage convention to a region whose indicator element indicates the second data storage convention, and in response to the recognition, to invoke the transition management software.

78. (original) The microprocessor chip of claim 77, wherein the microprocessor chip and software are designed to effect a transition between instruction boundaries, between execution on a page coded in the first instruction set using the first calling convention to execution on a page coded in the second instruction set using the second calling convention, the software programmed to effect the execution transition without the code at the source of the transition being specially coded to interface with code at the destination of the transition.

79. (original) The microprocessor chip of claim 77, wherein the two conventions are two calling conventions.

80. (original) The microprocessor chip of claim 79, wherein:

one of the two calling conventions is a register-based calling convention, and the other calling convention is a memory stack-based calling convention.

81. (original) The microprocessor chip of claim 79, wherein the physical resources of the microprocessor chip are associated to the logical resources of the first and second calling conventions according to a mapping that assigns corresponding logical resources to a common physical resource when the resources serve analogous functions in the two calling conventions.

82. (original) The microprocessor chip of claim 79, further comprising:

software and/or hardware designed to effect a transition between the execution of code coded under the first calling convention and code coded under the second calling convention, by altering a bit representation of a datum from a first representation under the first calling convention to a second representation under the second calling convention, the alteration of representation being chosen to preserve the meaning of the datum across the change in calling convention.

83. (original) The microprocessor chip of claim 79, further comprising:

software and/or hardware designed to copy a datum from a first location to a second location, the first location having a use under the first calling convention analogous to the use of the second location under the second calling convention.

84. (previously presented) The microprocessor chip of claim 79, further comprising:

software and/or hardware designed to copy a datum from a third location to a fourth, the third location having a use under the first data storage convention analogous to the use of the third location under the first data storage convention and to the fourth location under the second data storage convention, the software and/or hardware for the copying being

programmed to assume that exactly one of the first and third locations is no longer required by the execution of the program.

85. (last amended 7/12/2004) The microprocessor chip of claim 63, further comprising hardware and/or software designed:

(a) to retrieve calling convention indicator elements stored in association with respective pages of the memory, each calling convention indicator element indicating which of a register-based calling convention or a memory stack-based calling convention is observed by instructions of the page;

(b) to recognize when instruction execution has flowed or transferred from a page using the register-based calling convention to a page using the memory stack-based convention, as indicated by the calling convention indicator elements associated with the respective pages, and

(c) in response to the recognition, to alter a storage content of the computer to create a program context under the memory-based convention logically equivalent to a pre-alteration program context under the register-based convention.

86. (last amended 7/12/2004) The microprocessor chip of claim 63, wherein control-flow instructions of the microprocessor's instruction set are classified into a plurality of classes; and

the fetch and execute unit updates a record of the class of the classified control-flow instruction most recently executed;

the storage alteration process being determined, at least in part, by the instruction class record.

87. (last amended 4/25/05) A method, comprising the steps of:

executing a control-transfer instruction under a first execution mode of a computer, the instruction being architecturally defined to transfer control directly to a destination instruction for execution in a second execution mode of the computer;

before executing the destination instruction, altering the data storage content of the computer to establish a program context under the second execution mode that is logically equivalent to the context of the computer as interpreted under the first execution mode, the reconfiguring including at least one data movement operation not included in the architectural definition of the control-transfer instruction.

88. (original) The method of claim 87, wherein the data movement includes at least one of:

copying data from a general register to a memory stack; and

copying data from a memory stack to a general register.

89. (last amended 4/25/05) The method of claim 87, wherein:

the transition for the first execution mode to the second mode is recognized in a difference between two indicator elements associated with the memory pages containing the control-transfer instruction and the destination instruction, respectively, the pages being under the management of a virtual memory manager.

90. (previously presented) The method of claim 89, wherein the indicator elements are stored in a table whose entries are indexed by physical page frame number.

91. (original) The method of claim 89, wherein the indicator elements are stored in entries of a translation look-aside buffer.

92. (currently amended) The method of claim 87, further comprising the step of:

altering a bit representation of a datum from a first representation in the first mode to a second representation in the second mode, the alteration of representation being chosen to preserve the meaning of the datum across the change in execution mode.

93. (previously presented) The method of claim 87, wherein

a rule for copying data from the source memory or register to the destination register or memory is determined by examining a descriptor associated with the address of the control-transfer instruction.

94. (previously presented) A method, comprising the steps of:

executing a section of computer object code twice, without modification of the code section between the two executions, the code section materializing a destination address into a register and being architecturally defined to directly transfer control indirectly through the register to the destination address, the two executions materializing two different destination addresses;

the two destination code sections at the two materialized destination addresses being coded in two distinct instruction sets and, respectively, obeying the default calling conventions native to each of the two instruction sets, neither instruction set being a subset of the others.

95. (original) The method of claim 94, wherein:

the code at the first destination receives floating-point arguments and returns floating-point return values using a register-based calling convention; and

the code at the second destination receives floating-point arguments using a memory-based stack calling convention, and returns floating-point values using a register indicated by a top-of-stack pointer.

96. (previously presented) A microprocessor chip, comprising:

two instruction decoders designed to decode instructions of first and second instruction sets, respectively, and circuitry of a single instruction pipeline designed to execute the instructions decoded by either of the two instruction decoders;

circuitry and/or software designed to detect when execution flows or transfers control from code coded in one instruction set to code coded in the other, program code in the first and second instruction sets using first and second different data storage conventions, respectively; and

circuitry and/or software designed to respond to the detection by altering the data storage content of the computer to create a program context under the second data storage convention that is logically equivalent to a pre-alteration program context under the first data storage convention.

97. (previously presented) The computer processor of claim 96, wherein the memory unit and software are designed to effect a transition between instruction boundaries, between execution in a region coded in the first instruction set using the first data storage convention to execution in a region coded in the second instruction set using the second data storage convention, so that code at the source of the flow or transfer may effect the execution transition without being specially coded for code at the destination.

98. (previously presented) The microprocessor chip of claim 96, wherein the two data storage conventions are first and second calling conventions.

99. (previously presented) The computer processor of claim 98, wherein:

one of the two calling conventions is a register-based calling convention, and the other calling convention is a memory stack-based calling convention.

100. (previously presented) The microprocessor chip of claim 98, further comprising:

software and/or hardware designed to copy a datum from a first location to a second location, the first location having a use under the first calling conventions analogous to the use of the second location under the second calling conventions.

101. (previously presented) The microprocessor chip of claim 100, further comprising:

software and/or hardware designed to copy a datum from a third location to a fourth, the third location having a use under the first data storage convention analogous to the use of the third location under the first data storage convention and to the fourth location under the second data storage convention, the software and/or hardware for the copying being programmed to assume that exactly one of the first and third locations is no longer required by the execution of the program.

102. (previously presented) The computer processor of claim 98, wherein

a rule for altering the data storage content from the first calling convention to the second calling convention is determined based on an instruction at the location of execution at the source of the recognized execution flow or transfer.

103. (previously presented) The computer processor of claim 98, wherein

a rule for altering the data storage content from the first calling convention to the second calling convention is determined by examining a descriptor associated with the instruction before the recognized execution flow or transfer.

104. (last amended 4/25/05) A method, comprising the steps of:

executing instructions fetched from first, second and third regions of a single address space of the memory of a computer, the instructions of the first region being coded for

execution by computers of a first architecture and following a first data storage convention, the instructions of the second region being coded for execution by computers of a second architecture and following the first data storage convention, the instructions of the third region being coded for execution by computers of the second architecture and following a second data storage convention, the memory regions having associated modifiable indicator elements, a hardware structure for storing the indicator elements enforcing a requirement that the memory regions be necessarily disjoint, the modifiable indicator elements each having a value indicating the architecture and data storage convention under which instructions from the associated region are to be executed;

when execution of the instruction data flows or transfers between the first, second and third regions, adapting the computer for execution in the architecture and/or convention of the region transferred to.

105. (previously presented) The method of claim 104, wherein:

the regions are pages managed by a virtual memory manager.

106. (previously presented) The method of claim 105, wherein the modifiable indicator elements are stored in a table, each modifiable indicator element associated with a corresponding physical page frame.

107. (previously presented) The method of claim 105, wherein the entries are entries of a translation look-aside buffer.

108. (previously presented) The method of claim 104:

wherein the two architectures are two instruction set architectures;

and wherein the adapting step includes controlling instruction execution hardware of the computer to interpret the instructions according to the two instruction set architectures according to the modifiable indicator elements.

109. (previously presented) The method of claim 108, wherein:

one of the regions stores an off-the-shelf operating system binary coded in an instruction set non-native to the computer, the non-native instruction set providing access to a reduced subset of the resources of the computer.

110. (previously presented) The method of claim 108, wherein the two conventions are first and second data storage conventions, and further comprising the step of:

recognizing when program execution has flowed or transferred from a region whose modifiable indicator element indicates the first data storage convention to a region whose modifiable indicator element indicates the second data storage convention, and in response to the recognition, altering the data storage content of the computer to create a program context under the second data storage convention that is logically equivalent to a pre-alteration program context under the first data storage convention.

111. (previously presented) The method of claim 104, wherein the two conventions are first and second data storage conventions, and further comprising the step of:

recognizing when program execution has flowed or transferred from a region whose indicator element indicates the first data storage convention to a region whose indictor element indicates the second data storage convention, and in response to the recognition, altering the data storage content of the computer to create a program context under the second data storage convention that is logically equivalent to a pre-alteration program context under the first data storage convention.

112. (previously presented) The method of claim 111, wherein:

one of the two data storage conventions is a register-based calling convention, and the other data storage convention is a memory stack-based calling convention.

113. (last amended 7/12/2004) A computer processor, comprising:

a processor pipeline configured to alternately execute instructions of computers of two different architectures; and

a memory unit designed to fetch instructions from a computer memory for execution by the pipeline, and to fetch stored indicator elements associated with respective necessarily-disjoint memory regions of a single address space from which the instructions are to be fetched, each indicator element designed to store an indication of the architecture under which the instruction data of the associated region are to be executed by the processor pipeline, and to store a separate indicator element designed to indicate a data storage convention observed by instructions of the associated region;

the memory unit and/or processor pipeline further designed to recognize an execution flow or transfer from a region whose indicator element indicates one instruction set architecture to another; and

hardware and/or software designed to recognize when program execution has flowed or transferred from a region whose indicator element indicates the first data storage convention to a region whose indicator element indicates the second data storage convention, and in response to the recognition, to alter the data storage content of the computer to create a program context under the second data storage convention that is logically equivalent to a pre-alteration program context under the first data storage convention.

114. (previously presented) The computer processor of claim 113, wherein the two architectures are two instruction set architectures, and further comprising:

processor pipeline control circuitry designed to control the processor pipeline to effect interpretation of the instructions under the two instruction set architectures alternately, according to the associated indicator elements.

115. (previously presented) The computer processor of claim 114, further comprising:

software programmed to manage a transition between the execution of a program executing in the first instruction set architecture, being an instruction set architecture native to the computer processor, and execution of an off-the-shelf operating system coded in the second instruction set, being an instruction set non-native to the computer, providing access to a reduced subset of the resources of the computer.

116. (last amended 7/12/2004) The computer processor of claim 114:

the data storage convention indicator elements being designed to store an indication of a calling convention under which the instruction data of the associated region are coded for execution by the processor pipeline;

and further comprising software programmed to alter the data storage content of a computer using the computer processor to create a program context under the second calling convention that is logically equivalent to a pre-alteration program context under the first calling convention;

the memory unit further designed to recognize when program execution has flowed or transferred from a region whose indicator element indicates the first calling convention to a region whose indicator element indicates the second calling convention, and in response to the recognition, to invoke the transition management software.

117. (cancelled)

118. (cancelled)

119. (last amended 7/12/2004) The computer processor of claim 120, wherein the memory unit and software are designed to effect a transition between instruction boundaries, between execution in a region coded in the first instruction set using the first calling

convention to execution in a region coded in the second instruction set using the second calling convention, so that code at the source of the flow or transfer may effect the execution transition without being specially coded for code at the destination.

120. (last amended 7/12/2004) The computer processor of claim 116, wherein the two data storage conventions are two calling conventions.

121. (last amended 7/12/2004) The computer processor of claim 120, wherein:

one of the two calling conventions is a register-based calling convention, and the other calling convention is a memory stack-based calling convention.

122. (last amended 7/12/2004) The computer processor of claim 120, wherein the logical resources for support of the first and second instruction set architectures are overlaid on the physical resources of the computer processor according to a mapping that assigns corresponding resources of the two architectures to a common physical resource when the resources serve analogous functions in the calling conventions of the two architectures.

123. (last amended 7/12/2004) The computer processor of claim 120, further comprising:

a transition manager designed to effect a transition between execution under the first calling convention to execution under the second calling convention, the transition manager designed to alter a bit representation of a datum from a first representation to a second representation, the alteration of representation being chosen to preserve the meaning of the datum across the change in calling convention.

124. (last amended 7/12/2004) The computer processor of claim 120, further comprising:

software and/or hardware designed to copy a datum from a first location to a second location, the first location having a use under the first calling convention analogous to the use of the second location under the second calling convention.

125. (previously presented) The computer processor of claim 124, further comprising:

software and/or hardware designed to copy a datum from a third location to a fourth, the third location having a use under the first calling convention analogous to the use of the third location under the first calling convention and to the fourth location under the second calling convention, the software and/or hardware for the copying being programmed to assume that exactly one of the first and third locations is no longer required by the execution of the program.

126. (last amended 7/12/2004) The computer processor of claim 120, wherein

a rule for altering the data storage content from the first calling convention to the second is determined by examining a descriptor associated with the instruction before the recognized execution flow or transfer.

127. (last amended 7/12/2004) The computer processor of claim 120, wherein control-flow instructions of the instruction set are classified into a plurality of classes; and

the processor pipeline updates a record of the class of the classified control-flow instruction most recently executed;

the storage alteration process being determined, at least in part, by the instruction class record.

128. (previously presented) The computer processor of claim 113, wherein:

the regions are pages managed by a virtual memory manager.

129. (previously presented) The computer processor of claim 113, wherein the indicator elements are stored in virtual address translation table entries.

130. (previously presented) The computer processor of claim 113, wherein the indicator elements are stored in a table of indicator elements distinct from a primary address translation table used by the virtual memory manager, the indicator elements of the table associated with corresponding pages of the memory.

131. (previously presented) The computer processor of claim 113, wherein the indicator elements are stored in respective entries of a table whose entries are indexed by physical page frame number.

132. (previously presented) The computer processor of claim 113, wherein:

the indicator elements are stored in storage that is architecturally addressable when the processor pipeline is executing in one of the computer architectures or processing conventions, and architecturally unaddressable when the processor pipeline is executing in the other architecture or convention.

133. (previously presented) The computer processor of claim 113, further comprising circuitry designed to raise an exception when execution flows or transfers from a region whose indicator element indicates one architecture or execution convention to another.

134. (last amended 4/25/05) The method of claim 87, wherein the first and second execution modes are first and second instruction set architectures.

135. (last amended 4/25/05) The method of claim 87, wherein the first and second execution modes are first and second calling conventions.

136. (last amended 4/25/05) A method, comprising the steps of:

executing instructions fetched from first and second regions of a single address space of the memory of a computer, the instructions of the first and second regions being coded for execution by computers of first and second instruction sets and following first and second data storage conventions, the data storage conventions being established by software convention without being defined into the hardware definition of the first and second instruction sets, the first and second regions having respective associated modifiable indicator elements, the modifiable indicator elements each having a value indicating the instruction set under which instructions from the associated region are to be executed;

when the computer recognizes, based on the indicator elements, that execution has transferred, by flow or by a control transfer instruction, between the first and second regions, adapting at least a part of the data storage content of the computer from the data storage convention for the region transferred from to the data storage convention for the region transferred to before commencing execution in the transferred-to region, to create a program context under the data storage convention of the transferred-to region that is logically equivalent to a pre-alteration program context under the transferred-from data storage convention.

137. (last amended 4/25/05) The method of claim 136, wherein:

the regions are pages managed by a virtual memory manager.

138. (last amended 4/25/05) The method of claim 137, wherein the modifiable indicator elements are stored in a table, each modifiable indicator element associated with a corresponding physical page frame.

139. (last amended 4/25/05) The method of claim 137, wherein the entries are entries of a translation look-aside buffer.

140. (last amended 4/25/05) The method of claim 136, wherein:

one of the regions stores an off-the-shelf operating system binary coded in an instruction set non-native to the computer, the non-native instruction set providing access to a reduced subset of the resources of the computer.

141. (last amended 4/25/05) The method of claim 136, wherein:

one of the two data storage conventions is a register-based calling convention, and the other data storage conv

Evidence Appendix

Response to Office Action or Appeal Brief

CS152
Computer Architecture and Engineering

Lecture 5: Cost and Design

September 10, 1997

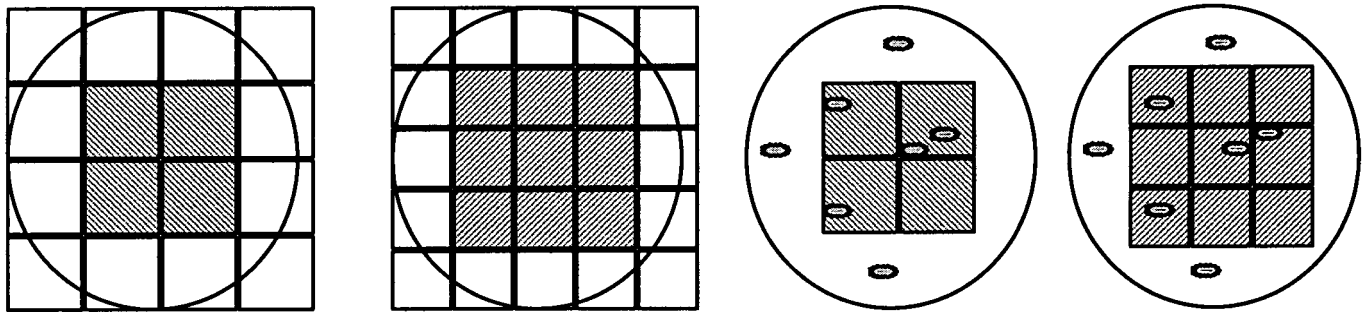
Dave Patterson (http.cs.berkeley.edu/~patterson)

lecture slides: http://www-inst.eecs.berkeley.edu/~cs152/

Integrated Circuit Costs

$$\text{Die cost} = \frac{\text{Wafer cost}}{\text{Dies per Wafer} * \text{Die yield}}$$

$$\text{Dies per wafer} \sim \text{eff} \frac{\text{Wafer Area}}{\text{Die Area}}$$



$$\text{Die Yield} = \frac{\text{Wafer yield}}{\left\{ 1 + \frac{\text{Defects\_per\_unit\_area} * \text{Die\_Area}}{?} \right\}^{?}}$$

Die Cost is goes roughly with the cube of the area.

Die Yield

Raw Dices Per Wafer

wafer diameter	*die area (mm^2)*					
	100	*144*	*196*	*256*	*324*	*400*
6"/15cm	139	90	62	44	32	23
8"/20cm	265	177	124	90	68	52
10"/25cm	431	290	206	153	116	90
die yield	23%	19%	16%	12%	11%	10%

typical CMOS process: ? =2, wafer yield=90%, defect density=2/cm2, 4 test sites/wafer

Good Dices Per Wafer (Before Testing!)

6"/15cm	31	16	9	5	3	2
8"/20cm	59	32	19	11	7	5
10"/25cm	96	53	32	20	13	9

typical cost of an 8", 4 metal layers, 0.5um CMOS wafer: ~$2000

Real World Examples

Chip	Metal layers	Line width	Wafer cost	Defect /cm^2	Area mm^2	Dies/ wafer	Yield	Die Cost
386DX	2	0.90	$900	1.0	43	360	71%	$4
486DX2	3	0.80	$1200	1.0	81	181	54%	$12
PowerPC 601	4	0.80	$1700	1.3	121	115	28%	$53
HP PA 7100	3	0.80	$1300	1.0	196	66	27%	$73
DEC Alpha	3	0.70	$1500	1.2	234	53	19%	$149
SuperSPARC	3	0.70	$1700	1.6	256	48	13%	$272
Pentium	3	0.80	$1500	1.5	296	40	9%	$417

From "Estimating IC Manufacturing Costs," by Linley Gwennap, *Microprocessor Report*, August 2, 1993, p. 15

Other Costs

$$\text{IC cost} = \frac{\text{Die cost} + \text{Testing cost} + \text{Packaging cost}}{\text{Final test yield}}$$

Packaging Cost: depends on pins, heat dissipation

Chip	*Die cost*	*Package pins*	*Package type*	*Package cost*	*Test & Assembly*	*Total*
386DX	$4	132	QFP	$1	$4	$9
486DX2	$12	168	PGA	$11	$12	$35
PowerPC 601	$53	304	QFP	$3	$21	$77
HP PA 7100	$73	504	PGA	$35	$16	$124
DEC Alpha	$149	431	PGA	$30	$23	$202
SuperSPARC	$272	293	PGA	$20	$34	$326
Pentium	$417	273	PGA	$19	$37	$473